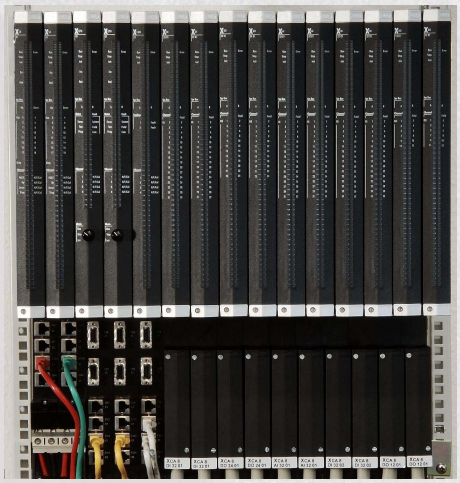


HIMax®

Communication Manual

SAFETY
NONSTOP

HIMA

COMMUNICATION

8

All HIMA products mentioned in this manual are protected by the HIMA trade-mark. Unless noted otherwise, this also applies to other manufacturers and their respective products referred to herein.

All of the instructions and technical specifications in this manual have been written with great care and effective quality assurance measures have been implemented to ensure their validity. For questions, please contact HIMA directly. HIMA appreciates any suggestion on which information should be included in the manual.

Equipment subject to change without notice. HIMA also reserves the right to modify the written material without prior notice.

For further information, refer to the CD-ROM and our website http://www.hima.de and http://www.hima.com.

Contact

HIMA contact details:

HIMA Paul Hildebrandt GmbH + Co KG

P.O. Box 1261

68777 Brühl, Germany

Tel: +49 6202 709-0

Fax: +49 6202 709-107

E-mail: info@hima.com

Table of Contents

1 Introduction

The communication manual describes the characteristics and configuration of the communication protocols of the safety-related HIMax control system.

Using the provided protocols, HIMax controllers can be connected with one another or with controllers from other manufacturers.

Knowledge of regulations and the proper technical implementation of the instructions detailed in this manual performed by qualified personnel are prerequisites for planning, engineering, programming, installing, starting up, operating and maintaining the HIMax automation devices.

HIMA will not be held liable for severe personal injuries, damage to property or the surroundings caused by any of the following: unqualified personnel working on or with the devices, de-activation or bypassing of safety functions, or failure to comply with the instructions detailed in this manual (resulting in faults or impaired safety functionality).

HIMax automation devices have been developed, manufactured and tested in compliance with the pertinent safety standards and regulations. They may only be used for the intended applications under the specified environmental conditions.

1.1 Structure and Use of this Manual

The content of this manual is part of the hardware description of the HIMax programmable electronic system.

This manual is organized in the following main chapters:

- Introduction
- Safety
- Product Description
- Start-up
- Operation
- Repairs
- Decommissioning
- Transport
- Disposal

Additionally, the following documents must be taken into account:

Name	Content	Document no.
HIMax System Manual	Hardware description of the HIMax system	HI 801 001 E
HIMax Safety Manual	Safety functions of the HIMax systems	HI 801 003 E
HIMax Communication Manual	Description of communication and protocols	HI 801 101 E
First Steps	Introduction to SILworX	HI 801 103 E

Table 1: Additional Valid Manuals

The latest manuals can be downloaded from the HIMA website www.hima.com. The revision index on the footer can be used to compare the current version of existing manuals with the Internet edition.

1.2 Target Audience

This document addresses system planners, configuration engineers, programmers of automation devices and personnel authorized to implement, operate and maintain the devices and systems. Specialized knowledge of safety-related automation systems is required.

1.3 Formatting Conventions

To ensure improved readability and comprehensibility, the following fonts are used in this document:

Bold:	To highlight important parts Names of buttons, menu functions and tabs that can be clicked and used in SILworX.
Italics:	For parameters and system variables
`Courier`	Literal user inputs
RUN	Operating state are designated by capitals
Chapter 1.2.3	Cross references are hyperlinks even though they are not particularly marked. When the cursor hovers over a hyperlink, it changes its shape. Click the hyperlink to jump to the corresponding position.

Safety notes and operating tips are particularly marked.

1.3.1 Safety Notes

The safety notes are represented as described below.
These notes must absolutely be observed to reduce the risk to a minimum. The content is structured as follows:

- Signal word: danger, warning, caution, notice
- Type and source of danger
- Consequences arising from the danger
- Danger prevention

⚠ SIGNAL WORD

Type and source of danger!
Consequences arising from the danger
Danger prevention

The signal words have the following meanings:

- Danger indicates hazardous situation which, if not avoided, will result in death or serious injury.
- Warning indicates hazardous situation which, if not avoided, could result in death or serious injury.
- Warning indicates hazardous situation which, if not avoided, could result in minor or modest injury.
- Notice indicates a hazardous situation which, if not avoided, could result in property damage.

NOTICE

Type and source of damage!

Damage prevention

1.3.2 Operating Tips

Additional information is structured as presented in the following example:

i The text corresponding to the additional information is located here.

Useful tips and tricks appear as follows:

TIP The tip text is located here.

2 Safety

The following safety information, notes and instructions must be strictly observed. The product may only be used if all guidelines and safety instructions are adhered to.

This product is operated in accordance with SELV or PELV. No imminent danger results from the module itself. The use in Ex-Zone is permitted if additional measures are taken.

2.1.1 Operating Requirements

The devices have been developed to meet the following standards for EMC, climatic and environmental requirements:

Standard	Content
EC/EN 61131-2	Programmable controllers, Part 2 Equipment requirements and tests
IEC/EN 61000-6-2	EMC Generic standards, Parts 6-2 Immunity for industrial environments
IEC/EN 61000-6-4	Electromagnetic Compatibility (EMC) Generic emission standard, industrial environments

Table 2: Standards for EMC, Climatic and Environmental Requirements

When using the safety-related HIMax control systems, the following general requirements must be met:

Requirement type	Requirement content
Protection class	Protection class II in accordance with IEC/EN 61131-2
Pollution	Pollution degree II in accordance with IEC/EN 61131-2
Altitude	< 2000 m
Enclosure	Standard: IP 20 If required by the relevant application standards (e.g., EN 60204, EN 954-1), the device must be installed in an enclosure of the specified protection class (e.g., IP 54).

Table 3: General requirements

Climatic Requirements

The following table lists the key tests and thresholds for climatic requirements:

IEC/EN 61131-2	Climatic tests
	Operating temperature: 0...+60 °C (test limits: -10...+70 °C)
	Storage temperature: -40...+85 °C
	Dry heat and cold resistance tests: +70 °C / -25 °C, 96 h, power supply not connected
	Temperature change, resistance and immunity test: -25 °C / +70 °C und 0 °C / +55 °C, power supply not connected
	Cyclic damp-heat withstand tests: +25 °C / +55 °C, 95 % relative humidity, power supply not connected

Table 4: Climatic Requirements

Mechanical Requirements

The following table lists the key tests and thresholds for mechanical requirements:

IEC/EN 61131-2	Mechanical tests
	Vibration immunity test: 5...9 Hz / 3.5 mm 9...150 Hz, 1 g, EUT in operation, 10 cycles per axis
	Shock immunity test: 15 g, 11 ms, EUT in operation, 2 cycles per axis

Table 5: Mechanical Tests

EMC Requirements

Higher interference levels are required for safety-related systems. HIMax systems meet these requirements in accordance with IEC 62061 and IEC 61326-3-1 (DIS). See column "Criterion FS“ (Functional Safety).

IEC/EN 61131-2	Interference immunity tests	Criterion FS
IEC/EN 61000-4-2	ESD test: 6 kV contact, 8 kV air discharge	-
IEC/EN 61000-4-3	RFI test (10 V/m): 26 MHz...1 GHz, 80 % AM RFI test (20 V/m): 26 MHz...2.7 GHz, 80 % AM: EN 298	- 20 V/M
IEC/EN 61000-4-4	Burst test: 2 kV power supply-, 1 kV signal lines	4 kV
IEC/EN 61000-4-12	Damped oscillatory wave test 2.5 kV L-,L+ / PE 1 kV L+ / L -	

Table 6: Interference Immunity Tests

IEC/EN 61000-6-2	Interference immunity tests	Criterion FS
IEC/EN 61000-4-6	High frequency, asymmetrical 10 V, 150 kHz...80 MHz, AM 20 V, 150 kHz...80 MHz, AM: EN 298	 20 V
IEC/EN 61000-4-3	900 MHz pulses	
IEC/EN 61000-4-5	Surge: 2 kV, 1 kV	2 kV / 1 kV

Table 7: Interference Immunity Tests

IEC/EN 61000-6-4	Noise emission tests
EN 55011 Class A	Emission test: radiated, conducted

Table 8: Noise Emission Tests

Power Supply

The following table lists the key tests and thresholds for the device's power supply:

IEC/EN 61131-2	Review of the DC supply characteristics
	Alternatively, the power supply must comply with the following standards: IEC/EN 61131-2 or SELV (Safety Extra Low Voltage) or PELV (Protective Extra Low Voltage)
	HIMax devices must be fuse protected as specified in this manual
	Voltage range test: 24 VDC, -20 %...+25 % (19.2 V...30.0 V)
	Momentary external current interruption immunity test: DC, PS 2: 10 ms
	Reversal of DC power supply polarity test: Refer to corresponding chapter of the system manual or data sheet of power supply.
	Backup duration withstand test: Test B, 1000 h

Table 9: Review of the DC Supply Characteristics

ESD Protective Measures

Only personnel with knowledge of ESD protective measures may modify or extend the system or replace a module.

NOTE

Electrostatic discharge can damage the electronic components within the controllers!

- When performing the work, make sure that the workspace is free of static and wear an ESD wrist strap.
- If not used, ensure that the module is protected from electrostatic discharge, e.g., by storing it in its packaging.

Only personnel with knowledge of ESD protective measures may modify or extend the system wiring.

2.2 Residual Risk

No imminent danger results from a HIMax module itself.

Residual risk may result from:

- Faults in the engineering
- Faults in the user program
- Faults in the wiring

2.3 Safety Precautions

Observe all local safety requirements and use the protective equipment required on site. Safety shoes are required while mounting the X-BASE PLATE.

2.4 Emergency Information

A HIMax controller is a part of the safety equipment of a system. If the controller fails, the system adopts the safe state.

In case of emergency, it is not permitted to access the safety equipment.

3 Product Description

The HIMax system is a safety-related control system and is intended for continuous operation and maximum availability.

HIMax is a modular system. Functions such as processing, input and output, and communication are distributed on plug-in modules.

In addition to using the physical input and output variables of the HIMax system, variables can also be exchanged with other system through a data connection.

The safety-related safe**ethernet** protocol and various standard protocols are available for exchanging process data.

3.1 Safety-Related Protocol (safeethernet)

All HIMax systems can safely communicate in SIL 3 via Ethernet. The safe**ethernet** protocol ensures safety-related communication.

The safety-related safe**ethernet** protocol is used to ensure that HIMax and HIMatrix controllers can safely exchange process data in an Ethernet network.

Processor modules:	safe**ethernet** is executed on the processor module. A maximum of 4 processor modules per HIMax controller.
Connections:	For each HIMax: a maximum of 255 safe**ethernet** connections. For each HIMatrix: a maximum of 64 safe**ethernet** connections.
Transmission paths:	Ethernet interfaces of the COM module and processor modules Ethernet interfaces in use can simultaneously be used for additional protocols.
Process data volume:	On each safe**ethernet** connection, a maximum of 1100 bytes for HIMax and 900 bytes for HIMatrix can be transferred in each direction.

i The maximum process data volume of the controller of 512 kB must not be exceeded. If this is the case, the controller configuration is rejected during the load process.

For more on the safety-related protocol safe**ethernet**, refer to Chapter 4.

⚠ WARNING

Manipulation of safety-related data transfer!

Physical injury

The operator is responsible for ensuring that the Ethernet used for safeethernet is sufficiently protected against manipulations (e.g., from hackers).
The type and extent of the measures must be agreed upon together with the responsible test authority.

3.2 Standard Protocols

Numerous proven standard protocols are available to ensure that field devices and control systems are optimally integrated in the HIMax systems. In this scenario, both Ethernet and field bus protocols can be used.

Many communication protocols only ensure a non-safety-related data transmission. These protocols can only be used for the non-safety-related aspects of an automation task.

Communication modules:	Standard protocols are run on the COM module
Transmission paths:	Ethernet interfaces and fieldbus interfaces of the COM.
Maximum number of standard protocols	HIMax: ▪ 20 COM modules per HIMax controller. ▪ 20 standard protocols per HIMax controllers. ▪ 6[1)] Standard protocols per COM module HIMatrix: ▪ 4 standard protocols per HIMatrix controllers.
Process data volume:	With all non-safety-related protocols, each HIMax controller can transmit a total of 128 bytes of data and receive a total of 128 bytes of data. each HIMatrix controller can transmit a total of 64 bytes of data and receive a total of 64 bytes of data.

[1)] SNTP client and SNTP server are not included in this calculation.

⚠ WARNING

Use of unsafe import data

Physical injury

Non-safe data must not be used for performing the safety functions of the user program.

The following standard protocols are available:

Protocol	Per module	Description
PROFINET IO Controller	1	Chapter 5.2
PROFINET IO Device	1	Chapter 5.7
PROFIBUS DP master	2	Chapter 6.1
PROFIBUS DP slave	1	Chapter 6.13
Modbus master	1	Chapter 7.1
Modbus slave	1	Chapter 7.9
S&R TCP	1	Chapter 8
HIMA X-OPC Server[1)]	---	Chapter 10
ComUserTask	1	Chapter 11

Table 10: Available Standard Protocols

[1)] The HIMA X-OPC server is installed on a host PC and is used as a transfer interface for up to 255 HIMax controllers and third-party systems that have an OPC interface.

i Max. 64 TCP connections per HIMatrix controller or HIMax COM module.
Max. 1280 TCP connections per HIMax controller with 20 COM module.

Maximum number of active protocols on one HIMatrix or one HIMax COM module

A maximum of 64 TCP sockets are available for each HIMatrix or HIMax COM module.

Example 1:

Protocol	Connections
1 Modbus master	TCP: 44 slave connections, RTU: 122 slave connections
1 Modbus slave	TCP: 20 master connections, RTU: 1 master connections

Table 11: Protocols on one Communication Module

Example 2:

Protocol	Connections
1 PROFIBUS DP master	122 slave connections
1 PROFIBUS DP slave	1 master connection

Table 12: Protocols on one Communication Module

3.3 Redundancy

The HIMax system conceptual design is characterized by high availability and also provides redundancy for the purpose of communication. A communication connection is redundant if two identical physical transmission paths exist.

Redundant communication on HIMatrix is only ensured via safe**ethernet**.

Redundancy via safeethernet

Redundancy is configured in the safe**ethernet** Editor by selecting the Ethernet interfaces for the two transmission paths (see Chapter 4.2).

Redundancy via Standard Protocols

PROFIBUS DP master PROFIBUS DP slave PROFINET IO TCP S&R Modbus master	Redundancy of the standard protocols must be configured in the user program such that the user program monitors the redundant transmission paths and assigns the redundantly transmitted process data to the corresponding transmission path.
Modbus slave	Redundancy can be set in SILworX.

3.4 Structure of the HIMax COM Module Part Number

The X-COM 01 module forms a functional unit with the X-CB 001 02 Connector Board. Note that the Connector Board must be separately purchased.

When the X-COM 01 is equipped with one or several fieldbus submodules, the module name changes to X-COM 010 XY*, see Table 13.

The available components and their part numbers are listed below:

Designation	Description	Part no.
X-COM 01	Communication module without fieldbus submodule	98 5260000
X-COM 010 **XY**	Communication module with fieldbus submodule	98 52600**XY**
X-CB 001 02	Connector board	98 5020008

Table 13: Part Numbers

x: Option for fieldbus submodule 1 (fieldbus interface 1)

y: Option for fieldbus submodule 2 (fieldbus interface 2)

Option values for **x** and **y**:

0: No fieldbus submodule inserted

1: RS485 (Modbus master, Modbus slave, depending on the license code) or CUT (ComUserTask)

2: PROFIBUS DP master

3: PROFIBUS DP slave

4 ---

5 RS232 submodule for CUT (ComUserTask) inserted.

6 RS422 submodule for CUT (ComUserTask) inserted.

The part number (Part no.) is printed on the type label of the module.

Examples:

Part no.	Fieldbus submodule 1 (FB1)	Fieldbus submodule 2 (FB2)
98 52600 21	PROFIBUS master	RS485
98 52600 23	PROFIBUS master (max. 12 Mbit)	PROFIBUS slave (max. 1.5 Mbit)
98 52600 11	RS485	RS485
98 52600 00	-	-

Table 14: Examples of COM Module Part Numbers

i HIMA recommends operating the PROFIBUS DP using the FB1 fieldbus interface (maximum transfer rate 12 Mbit). The maximum transfer rate permitted for the FB2 fieldbus interface is 1.5 Mbit.

⚠ CAUTION

Improper opening of the COM module

Damage to COM module

Only HIMA may mount fieldbus submodules on the communication module.

3.5 Protocol Registration and Activation

The protocols specified below are available for the HIMax systems and can be activated as follows:

Protocol	Interfaces	Activation
HIMA safe**ethernet**	Ethernet	[1]
HIMA X-OPC server (it runs on a the host PC)	Ethernet	[4]
Modbus TCP master	Ethernet	[4]
Modbus TCP slave	Ethernet	[4]
TCP send/receive	Ethernet	[4]
SNTP server/client	Ethernet	[4]
PROFIBUS DP master	FB1 and FB2	[2]
PROFIBUS DP slave	FB1 or FB2	[2]
Modbus master RS485	FB1 or FB2	[3]
Modbus slave RS485	FB1 or FB2	[3]
CUT (ComUserTask) RS232 and RS422	FB1 or FB2	[3]

Table 15: Protocols Available for the HIMax Systems

[1]. safe**ethernet** is activated by default in all HIMax systems.

[2]. The PROFIBUS master and PROFIBUS slave are activated by installing one fieldbus submodule.

[3]. Additionally, a software activation code must be purchased for the selected fieldbus protocol used with the RS485 fieldbus submodule (Modbus RS485), and with the RS232 and RS422 fieldbus submodules (ComUserTask).

[4]. The software activation code can be generated on HIMA website using the system ID of the controller. Follow the instructions provided on the HIMA website.

i The software activation code unseparately connected with this system ID. One license can only be used one time for a specific system ID. For this reason, only activate the code when the system ID has been uniquely defined.

All Ethernet protocols can be tested without software activation code for 5000 operating hours .

i To avoid potential delays, remember to generate your software activation code on time! After these 5000 operating hours, communication is continued until the controller is stopped. Afterwards, the user program cannot be started without a valid software activation code for the protocols used in the project (invalid configuration).

To enter the software activation code in SILworX

1. In the structure tree, select **Configuration, Resource, License Management**.
2. Right-click **License Management,** and then select **New, License Key**.
   ☑ A new license key is created.
3. Right-click **License Key**, then click **Properties**.
4. Enter the new software activation code in the **Activation Code** field.

3.6 Ethernet Interfaces

The Ethernet interfaces of the processor and COM modules are used to communicate with external systems. Each individual Ethernet interface can simultaneously process several protocols.

i Process data cannot be transferred over the Ethernet interface of the X-SB 01 system bus module. The UP and DOWN Ethernet interfaces are exclusively intended for interconnecting HIMax base plates.

3.6.1 Ethernet Interfaces Properties

Property	Processor module	HIMax COM module HIMatrix
Ports	4	4
Transfer standard	10/100/1000 Base-T, half and full duplex	10/100 Base-T, Half and full duplex
Auto negotiation	Yes	Yes
Auto crossover	Yes	Yes
Connection socket	RJ-45	RJ-45
IP address	Freely configurable[1)]	Freely configurable[1)]
Subnet mask	Freely configurable[1)]	Freely configurable[1)]
Supported protocols	safe**ethernet** Programming and debugging tool (PADT) SNTP	safe**ethernet** Programming and debugging tool (PADT) TCP SR, SNTP, Modbus TCP/UDP

Table 16: Ethernet Interfaces Properties

[1)] The general rules valid for assigning IP address and subnet masks must be adhered to.

Each COM and processor module is equipped with one Ethernet switch with IP address and four ports on the corresponding connector board.

To transfer data, the Ethernet switch of one processor or COM module establishes a targeted connection between two communication partners. This prevents collisions and reduces the load on the network.

For targeted data forwarding, a MAC/IP address assignment table (ARP cache) is generated in which the MAC addresses are assigned to specific IP addresses. From now on, data packets are only forwarded to the IP addresses specified in the ARP cache.

i Replacement of one processor module or one COM module with identical IP address
If a processor module or COM module has its ARP Aging time set to 5 minutes and its MAC Learning set to Conservative, its communication partner does not adopt the new MAC address until a period of 5 to 10 minutes after the module is replaced. Until the new MAC address has been adopted, no communication is possible using the replaced processor module or COM module.

In addition to the configurable ARP Aging time, the user must wait at least the non-configurable MAC Aging time of the switch (approx. 10 seconds) before the replaced processor or COM module is able to communicate again.

3.6.2 Configuring the Ethernet Interfaces

The Ethernet interfaces are configured in the Detail View of the processor or COM module.

For HIMax controllers, the *Speed Mode* and *Flow-Control Mode* parameters are set per default to Autoneg.

Communication lost!

With an inappropriate Ethernet parameters setting, the processor or communication module might no longer be reachable. If this is the case, reset the module!

To open the Detail View of the communication module

1. In the structure tree, open **Configuration, Resource, Hardware**.
2. Right-click **Hardware ,** and then click **Edit** to open the Hardware Editor.
3. Right-click **Communication Module** , and then click **Detail View.** The Detail View opens.

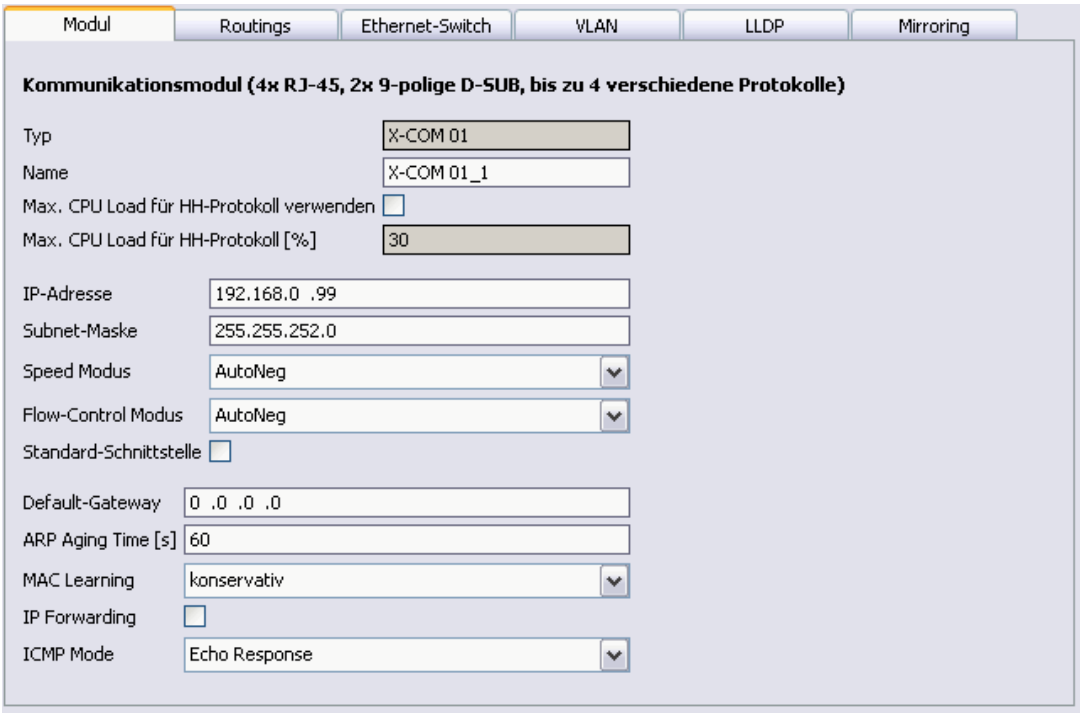


Figure 1: Dialog Box for Configuring the Processor and COM Modules in SILworX

The parameters set in the properties of the COM or the processor modules are not available for the HIMax system communication, until they have been re-compiled with the user program and transferred to the controller.

Module

Element	Description
Name	Name of the communication module.
Use Max CPU Load for HH Protocol	▪ Activated: Use CPU load limit from the *Max. CPU Load [%]* field. ▪ Deactivated: Do not use the CPU Load limit for safe**ethernet.**
Max. CPU Load for HH Protocol [%]	Maximum CPU load of module that can be used for processing the safe**ethernet** protocols. i The maximum load must be distributed among all the implemented protocols that use this communication module.
IP Address	IP address of the Ethernet interface.
Subnet Mask	32 bit address mask to split up the IP address in network and host address.
Speed Mode Flow Control Mode	i Only the Autoneg setting is permitted! With other settings, the module adopts the STOP state.
Standard Interface	Activated: the interface is used as standard interface for the system login.
Default Gateway	IP address of the default gateway.
Activate Extended Settings	Use the *ARP Aging Time [s], MAC Learning* and *IP Forwarding* parameters.
ARP Aging Time [s]	A processor or COM module stores the MAC addresses of the communication partners in a MAC/IP address assignment table (ARP cache). If in a period of 1 to 2 time *ARP Aging Time ...* - ... messages of the communication are received, the MAC address remains stored in the ARP cache. - ... no messages of the communication partner are received, the MAC address is erased from the ARP cache. The typical value for the *ARP Aging Time* in a local network ranges from 5...300 s. The user cannot read the contents of the ARP cache. If routers or gateways are used, the user must adjust (increase) the *ARP Aging Time* due to the additional time required for two-way transmission. If the *ARP Aging Time* is too low, the processor or the COM module deletes the MAC address of the communication partner from the ARP cache and the communication is either delayed or breaks down entirely. For an efficient performance, the ARP aging time value must be less than the receive timeout set for the protocols in use. Range of values: 1...3600 s Default value: 60 s

MAC Learning	MAC Learning and *ARP Aging Time* are used to set how quick the Ethernet switch should learn the MAC address. The following settings can be configured: ▪ Conservative (recommended): If the ARP cache already contains MAC addresses of communication partners, these are locked and cannot be replaced by other MAC addresses for at least one *ARP Aging Time* and a maximum of two *ARP Aging Time* periods. This ensures that data packets cannot be intentionally or unintentionally forwarded to external network participants (ARP spoofing). ▪ Tolerant: When a message is received, the IP address contained in the message is compared to the data in the ARP cache and the MAC address stored in the ARP cache is immediately overwritten with the MAC address from the message. Use the Tolerant setting if the availability of communication is more important than the authorized access to the controller.
IP Forwarding	Allow a processor or COM module to operate as router and to forward data packets to other network nodes. ▪ Activated: Forwarding is enabled. ▪ Deactivated: Forwarding is disabled.
ICMP Mode	The Internet Control Message Protocol (ICMP) allows the higher protocol layers to detect error states on the network layer and optimize the transmission of data packets. Message types of Internet Control Message Protocol (ICMP) supported by the processor module: ▪ No ICMP Responses All the ICMP commands are deactivated. This ensures a high degree of safety against potential sabotage that might occur over the network. ▪ Echo Response If Echo Response is activated, the node responds to a ping command. It is thus possible to determine if a node can be reached. Safety is still high. ▪ Host unreachable Not important for the user. Only used for testing at the manufacturer's facility. ▪ All Implemented ICMP Responses All ICMP commands are activated. This allows a more detailed diagnosis of network malfunctions.

Table 17: Configuration Parameters

Routings

Element	Description
Name	Denomination of the routing settings
IP Address	Target IP address of the communication partner (with direct host routing) or network address (with subnet routing). Range of values: 0.0.0.0 ... 255.255.255.255 Default value: 0.0.0.0
Subnet Mask	Define the target address range for a routing entry. 255.255.255.255 (with direct host routing) or subnet mask of the addressed subnet. Range of values: 0.0.0.0 ... 255.255.255.255 Default value: 255.255.255.255
Gateway	IP address of the gateway to the addressed network. Range of values: 0.0.0.0 ... 255.255.255.255 Default value: 0.0.0.1

Table 18: Routing Parameters

Ethernet Switch

Element	Description
Port	Port number as printed on the enclosure; per port, only one configuration may exist. Range of values: 1... 4
Speed [Mbit/s]	10 Mbit/s: Data rate 10 Mbit/s 100 Mbit/s: Data rate 100 Mbit/s 1000 Mbit/s: Data rate 1000 Mbit/s (processor module) Autoneg (10/100/1000): Automatic baud rate setting Default value: Autoneg
Flow Control	Full duplex: Simultaneous communication in both directions Half duplex: Communication in one direction Autoneg: Automatic communication control Default value: Autoneg
Autoneg also with Fixed Values	The "Advertising" function (forwarding the speed and flow control properties) is also performed if the parameters Speed and Flow Control have fixed values. This allows other devices with ports set to *Autoneg* to recognize the HIMax port settings.
Limit	Limit the inbound multicast and/or broadcast packets.OFF Off: No limitation Broadcast: Limit broadcast packets (128 kbit/s) Multicast and Broadcast: Limit multicast and broadcast packets (1024 kbit/s) Default value: Broadcast

Table 19: Ethernet Switch Parameters

VLAN (Port-Based VLAN)

For configuring the use of port-based VLAN.

i Should VLAN be supported, port-based VLAN should be off to enable each port to communicate with the other switch ports.

For each switch port, the user can define which switch other port received Ethernet frames may be sent to.
The table in the VLAN tab contains entries through which the connection between two ports can be set as active or inactive.

	Eth1	Eth2	Eth3	Eth4
Eth1				
Eth2	Active			
Eth3	Active	Active		
Eth4	Active	Active	Active	
COM	Active	Active	Active	Active

Table 20: VLAN Tab

Default setting: All connection between ports active

LLDP

LLDP (Link Layer Discovery Protocol) allows the use of the own device to send information (such as MAC address, device name, port number) per multicast in periodic intervals and to receive the same information from the devices closed-by.
The processor and communication modules support LLDP on the Eth1, Eth2, Eth3 and Eth4 ports.

The following parameters define how a given port should work:

Off	LLDP is disabled on this port
Send	LLDP sends LLDP Ethernet frames, received LLDP Ethernet frames are deleted without being processed.
Receive	LLDP sends no LLDP Ethernet frames, but received LLDP Ethernet frames are processed.
Send/Receive	LLDP sends and processes received LLDP Ethernet frames.

Default setting: Send/Receive

Mirroring

Mirroring is used to configure wether the module should duplicate Ethernet packets on a given port such that they can be read from a device connected to that port, e.g., for test purposes.

The following parameters define how a given port should work:

Off	This port does not participate in mirroring.
Egress:	Outgoing data of this port are duplicated.
Ingress:	Incoming data of this port are duplicated.
Ingress/Egress:	Incoming and outgoing data of this port are duplicated.
Dest Port:	This port is used to send duplicated data.

Default setting: OFF

3.6.3 Network Ports Used for Ethernet Communication

UDP Ports / Use

8000	Programming and operation with SILworX
8001	PES used to configure the remote I/Os
6010	safeethernet and OPC
123	SNTP (time synchronization between PES and remote I/O, PES and external devices)
8895	Modbus master UDP, if configured
502	Modbus salve (can be modified by the user)

TCP Ports / Use

8895	Modbus master
502	Modbus salve (can be modified by the user)
Xxx	TCP SR assigned by the user

3.7 Fieldbus Interfaces

Fieldbus interfaces of the COM module can be used to communicate with external systems. Only one protocol can be operated on each single fieldbus interface.

The fieldbus interfaces must be equipped with a fieldbus submodule. If no fieldbus submodule is used, communication is not possible on this interface. The transfer standard for the interface depends on the fieldbus submodule.

Designation	Interface	Protocol
FB 1	D-sub connector 9-pole	PROFIBUS master PROFIBUS slave Modbus master RS485 Modbus slave RS485 ComUserTask (RS232)
FB 2	D-sub connector 9-pole	PROFIBUS master PROFIBUS slave Modbus master RS485 Modbus slave RS485 ComUserTask (RS232)

Table 21: Fieldbus Interfaces

3.7.1 Pin Assignment of D-SUB Connectors FB1 and FB2

i The PIN assignment of the fieldbus interfaces depends on the fieldbus submodule used and is specified in the following table.

Fieldbus Submodule for PROFIBUS DP Master or Slave

Pin	Signal	Function
1	---	---
2	---	---
3	RxD/TxD-A	Receive/send Data A
4	RTS	Control signal
5	DGND	Data reference potential
6	VP	+5 V supply voltage
7	---	---
8	RxD/TxD-B	Receive/send Data B
9	---	---

Table 22: Pin Assignment of D-Sub Connectors FB1 and FB2 for PROFIBUS DP

RS485 Fieldbus Submodule for Modbus Master or Slave

Pin	Signal	Function
1	---	---
2	RP	+5 V decoupled with diodes
3	RxD/TxD-A	Receive/send Data A
4	CNTR-A	Control signal A
5	DGND	Data reference potential
6	VP	+5 V supply voltage
7	---	---
8	RxD/TxD-B	Receive/send Data B
9	CNTR-B	Control signal B

Table 23: Pin Assignment of D-Sub Connectors FB1 and FB2 for Modbus

Fieldbus Submodule RS232 for ComUserTask

Pin	Signal	Function
1	---	---
2	TxD	Send Data
3	RxD	Received data
4	---	---
5	DGND	Data reference potential
6	---	---
7	RTS	Requirement to be sent
8	---	---
9	---	---

Table 24: Pin Assignment of D-Sub Connectors FB1 and FB2 for RS232

4 safeethernet

All HIMax systems are safe**ethernet** capable. They can safely communicate in SIL 3 via Ethernet (HIMax 1 Gbit/s , HIMatrix 100 Mbit/s)

The corresponding Ethernet interfaces of the HIMax controllers can be also used for other protocols.

Various Ethernet network topologies can be used to ensure safe**ethernet** communication between controllers. To improve the data transfer speed and efficiency, tailor the safe**ethernet** protocol parameters to the Ethernet network in use.

These parameters can be set using so-called network profiles. The factory parameter settings ensure communication even if the user is not thoroughly familiarized with the network configuration.

i The safe**ethernet** protocol is safety-related and certified by the TÜV up to SIL 3 in accordance with IEC 61508.

Equipment and System Requirements:

HIMA controller	HIMax with processor module
Activation	This function is activated by default in all HIMax systems.

safeethernet (Properties):

Processor modules	safe**ethernet** is executed on the processor module. A maximum of 4 processor modules per HIMax controller.
Connections	For each HIMax: a maximum of 255 safe**ethernet** connections. For each HIMatrix: a maximum of 64 safe**ethernet** connections.
Transmission path	Ethernet interfaces of the COM module and processor modules. Ethernet interfaces in use can simultaneously be used for additional protocols.
Redundant transmission paths	Two-channel operation Redundant safe**ethernet** transmission paths between HIMax and HIMax systems can be configured in the safe**ethernet** Editor.
Process data volume	On each safe**ethernet** connection, a maximum of 1100 bytes for HIMax and 900 bytes for HIMatrix can be transferred in each direction. i The maximum process data volume of the controller of 512 kB must not be exceeded. If this is the case, the controller configuration is rejected during the load process.
Cross-project communication	safe**ethernet** connections to a resource in another project can be configured in SILworX, see Chapter 4.8.

4.1 What is safeethernet?

Requirements as determinism, reliability, interchangeability, extendibility and above all safety, are central issues within the process and automation technology.

safe**ethernet** is a transfer protocol for transmitting safety-related data up to SIL 3 when Ethernet technology is used.

safe**ethernet** implements mechanism that can detect and safely react to the following faults:

- Corruptions of the transmitted data (duplicated, lost and changed bits)
- Wrong message addressing (transmitter, receiver)
- Wrong data sequence (repetition, lost, change)
- Wrong timing (delay, echo)

safe**ethernet** is based on the IEEE 802.3 standard.

The standard Ethernet protocol frame is used to transmit safety-related data.

safe**ethernet** uses "unsafe data transfer channels" (Ethernet) in accordance with the "black channel" approach and monitors them on the transmitter and receiver side by using safety-related protocol mechanism. This allows the user to use normal Ethernet network components such as hubs, switches, routers within a safety-related network.

The significant difference to standard Ethernet is the safety and the real-time ability of safe**ethernet**. A special protocol mechanism ensures a deterministic behavior even if faults occur or new communication participants join the network. The system automatically integrates new components in the running system. All network components can be replaced during operation. Transmission times can be clearly defined using switches. Ethernet is thus real-time capable.

The possible transfer speed up to 1 Gbit/s for safety-related data is higher than the speed normally used. Transmission media such as copper lines and fiber optic cables can be used.

safe**ethernet** allows both connections to the company Intranet and to the Internet. Therefore, just one network is required for both safe and unsafe data transmission.

i The network may be shared with other participants if sufficient transfer capacity is available.

safe**ethernet** allows the user to create flexible system structures for decentralize automation with defined reaction times. Depending on the requirements, the intelligence can be distributed to the network participants in a centralized or decentralized manner.

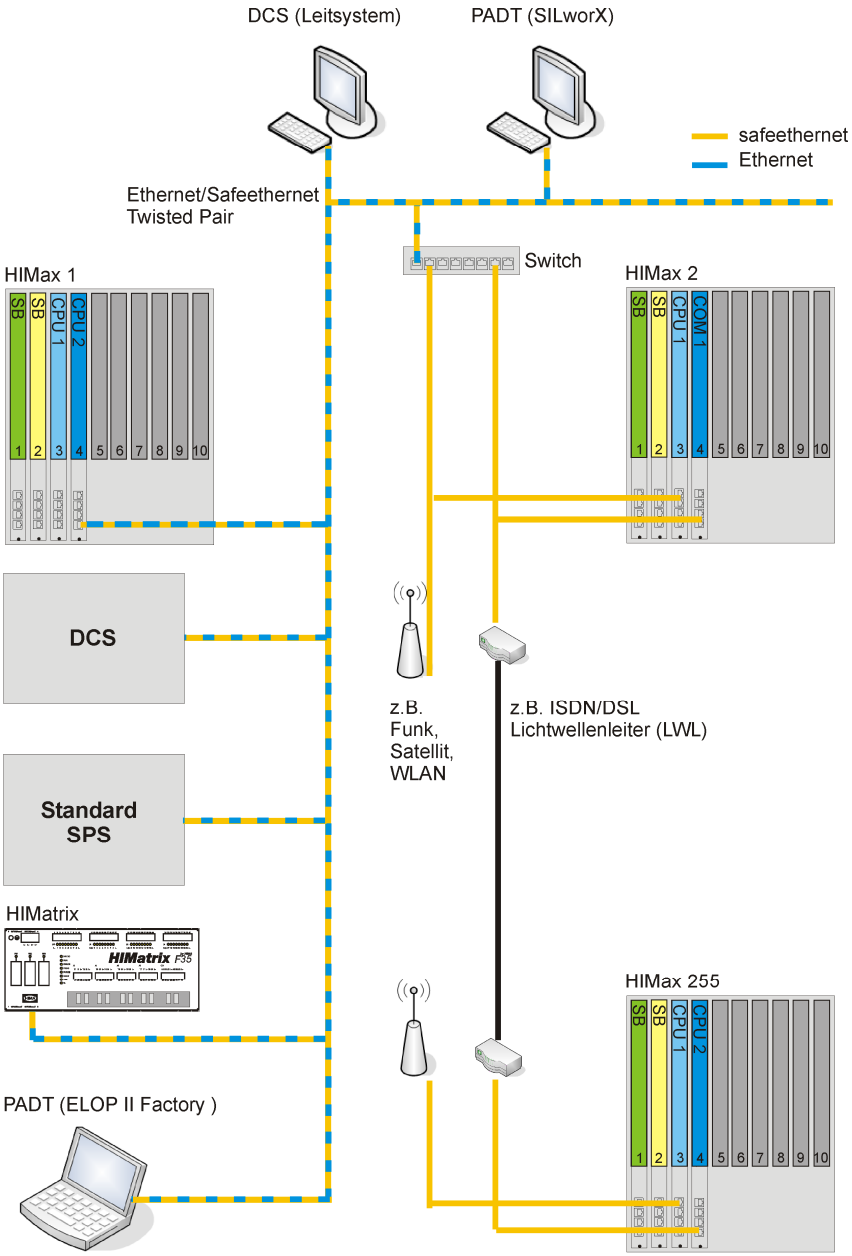


Figure 2: System Structures

i

Unintentional transition to the safe state possible!

Ensure that no network loops result from interconnecting the controllers. Data packets may only travel to a controller over a single path.

4.2 Configuring a Redundant safeethernet Connection

This example shows how to configure a redundant HIMax/HIMax safe**ethernet** connection.

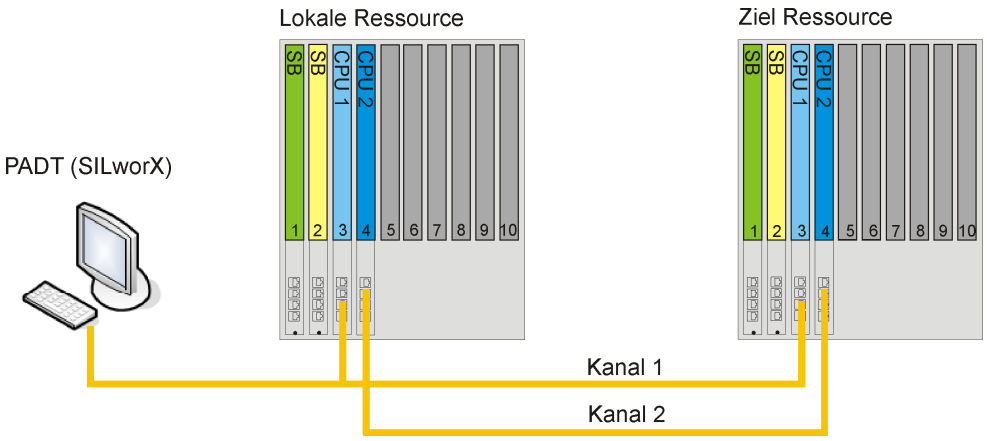


Figure 3: Structure for Configuring a Redundant Connection

i For redundant safe**ethernet** connections, HIMA recommends using two separate communication modules to implement the two transmission paths (channel 1 and channel 2). When doing so, the bandwidth and the delay on the respective transmission paths must be nearly identical.

Establishing the safeethernet Connection

In the safe**ethernet** Editor, create a safe**ethernet** connection between the locale and the target resource.

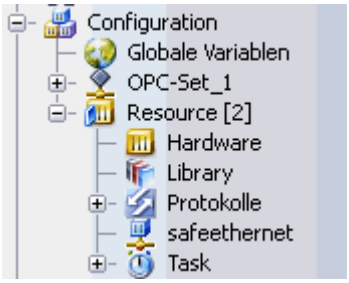


Figure 4: Resource Structure Tree

To open the safeethernet Editor of the local resource

1. In the structure tree, open **Configuration, Resource.**
2. Right-click safe**ethernet,** then click **Edit**.

   ☑ The target resources are located in the Object Panel.

To create the safeethernet connection to the target resource

1. In the Object Panel, drag the **target resource** anywhere in the workspace of the safe**ethernet** Editor.

i The reciprocal communication path is automatically added in the safe**ethernet** Editor of the target resource.

To configure the safeethernet connection

1. Select **Ethernet interfaces Channel 1** on the local and target resource.
2. Select **Ethernet interfaces Channel 2** on the local and target resource.
3. Select the **Network Profile** for the safe**ethernet** connection (e.g., Fast&Noisy)..
4. Calculate and enter **Receive Timeout** and **Response Time** (see Chapter 4.6).

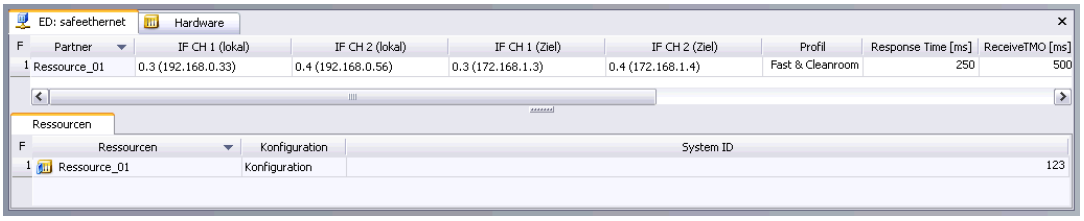


Figure 5: Parameter Values for a safe**ethernet** Connection:

Connecting Process Variables

To open the Detail View of a safeethernet connection

Requirement: The safe**ethernet** Editor of the local resource is open.

1. Right-click the **Target Resource** to open the context menu.
2. Select **Detail View**.
3. Select the **Resource (target)<->Resource (local)** tab.

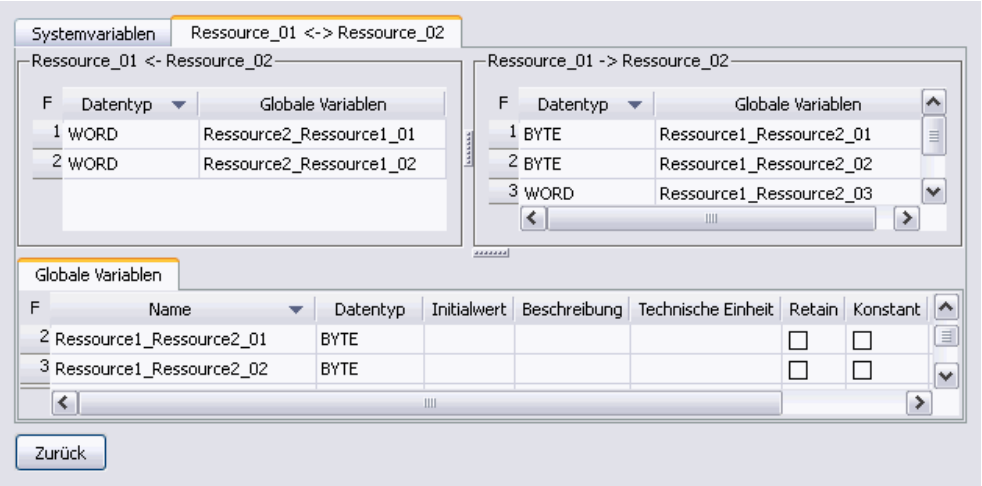


Figure 6: Detail View in the safe**ethernet** Editor

i Only global variables from the configuration context may be used, and not from the resource context!

To add safeethernet send variables

Send variables are sent from the local to the target resource.

1. Select the **Resource (local)->Resource (target)** area.
2. In the Object Panel, select a **Global Variable** and drag it onto the **Resource (target)->Resource (local)** column.
3. Repeat these steps for every further safe**ethernet** send variables.

To add safeethernet receive variables

Receive variables are received by the local resource.

1. Select the **Resource (target) <-Resource (local)** area.
2. In the Object Panel, select a **Global Variable** and drag it onto the **Resource (target)<-Resource (local)** column.
3. Repeat these steps for every further safe**ethernet** receive variables.

To verify the safeethernet connection

1. In the structure tree, open **Configuration, Resource, safeethernet**.
2. Click the **Verification** button on Action Bar, and then click **OK** to confirm the action.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

i The configuration of the safe**ethernet** connection must be recompiled with the user program of the local and target resources and transferred to the controllers. The new configuration can only be used for communicating with the HIMax system after this step is completed.

Verifying safeethernet Communication

In the Control Panel, reset the *Wrong Messages* and *Resends* values to zero.

1. Use the system under the maximum load:
   - All communication protocols are operating (safe**ethernet** and standard protocols).
   - Remove and re-insert the processor module such as described in Chapter 4.6.1.
   - Perform a reload to load the user program.

i To verify that the redundant safe**ethernet** connection was established properly, disconnect and reconnect one redundant connection and then repeat this test for the other connections. During this test, no faults must occur in the safe**ethernet** communication.

2. In the Control Panels of the two controllers, verify the *Wrong Messages* and *Resends* values. If the counter for *Bad Messages* and *Resends*
   = 0, then the safe**ethernet** settings are OK.
   ≥ 0, the safe**ethernet** settings must be rechecked.
   - Recalculate the *Receive Timeout* using the maximum cycle time, see Chapters 4.6.1 and 4.6.2.
   - Vary the *Response Time* such as described in Chapter 4.6.3.

i Additional causes for *bad messages* and *resends!*

Verify the correct network design (e.g., lines, switches, PCs). If the Ethernet network is not exclusively used for safe**ethernet**, also verify the network load (probable data collisions).

4.3 safeethernet Editor

The safe**ethernet** Editor is used to create and configure the safe**ethernet** connections to the communication partners (resources).

To open the safeethernet Editor of the local resource

1. In the structure tree, open **Configuration, Resource.**
2. Right-click safe**ethernet,** then click **Edit**.
   ☑ The safe**ethernet** Editor includes the workspace and the Object Panel

The safe**ethernet** Editor is used to create and configure the safe**ethernet** connections to the communication partners (resources). To do this, drag the resource from the Object Panel onto the workspace.

Set the following safe**ethernet** protocol parameters to configure the safe**ethernet** connection:

Parameter	Description
Partner	Resource name of the link partner
IF CH...	Ethernet interfaces available on the (local) and (target) resource, see also Chapter 3.6.
Profile	Combination of matching safe**ethernet** parameters, see also Chapter 4.7.5.
Sync/Async	Set the data exchange synchronous or asynchronous to the CPU cycle, see Chapter 4.6.4.
Response Time [ms]	Time until the acknowledgment of a message is received by the sender, see also Chapter 4.6.3.
Receive Timeout [ms]	Monitoring time of PES 1 within which a correct response from PES 2 must be received, see also Chapter 4.6.2.
Resend Timeout [ms]	Monitoring time of PES1 within which PES2 must have acknowledged the reception of a data packet, otherwise the data packet is resent, see also Chapter 4.6.5.
Acknowledge Timeout [ms]	Time period within which the CPU must acknowledge the reception of a data packet, see also Chapter 4.6.6.
Prod. Rate	Production rate: Minimum time interval between two data packets, see also Chapter 4.6.7.
Queue	Number of data packets that can be sent without acknowledgment, see also Chapter 4.6.8.

Freeze Values on Lost Connection [ms]	Behavior of the input variables for this safe**ethernet** connection if the connection is interrupted. Use initial values: The initial data are used for the input variables. Not limited: The input variables are freezed to the current value and used until a new connection is established. Limited: Input: Double-click on drop-down field and enter time. The input variables are freezed to the current value and used until the configured timeout. Afterwards the initial data are used for the input variables. The start of timeout can be delayed up to one CPU cycle. **⚠ CAUTION**  **For safety-related functions implemented via safeethernet, only the *Use Initial Data* setting may be used.**
Fragments per Cycle	Fixed setting: One view per controller cycle is transferred to the communication partner. The view of a HIMax is a fragment of ≤ 1100 bytes. The view of a HIMatrix is a fragment of ≤ 900 bytes.
Ignored Warning Count	No function yet!
Warning period [ms]	No function yet!
Enable SOE	Default value: activated

Table 25: safe**ethernet** Protocol Parameters

Object Panel

The Object Panel contains all the project resources with which the current resource can be connected via safe**ethernet**.

i An export function is available to establish safe**ethernet** connections to resources outside the project or to a HIMatrix controller (projected in ELOP II Factory) (see Chapter 4.8).

4.4 Detail View of the safeethernet Editor

The **Detail View** has always a reference to the local resource for which the safe**ethernet** Editor was started.

To open the Detail View of a safeethernet connection

1. Right-click the safe**ethernet** connection to open the context menu.
2. Click **Detail View**.
   ☑ The **Detail View** contains the **System Variables, View Definitions** and **Resource (local)<->Resource (target)** tabs.

4.4.1 Tab: System Variables

System variables are used to control the safe**ethernet** connection in the user program and evaluate its status.

<table>
<tr><th>System Variables</th><th>Description</th></tr>
<tr><td>Ack. Frame No.</td><td>Reception Counter (rotation)</td></tr>
<tr><td>Number of bad messages</td><td rowspan="2">Number of all the bad messages per channel (invalid CRC, invalid header, other faults)</td></tr>
<tr><td>Number of bad messages for the red. channel</td></tr>
<tr><td>Number of successful connections</td><td>Number of successful connections since statistics reset.</td></tr>
<tr><td>Number of lost messages</td><td rowspan="2">Number of messages dropped out on one of the two transmission paths since statistics reset.
The counter only continues to run until a channel completely fails.</td></tr>
<tr><td>Number of lost messages for the red. channel</td></tr>
<tr><td>Early Queue Usage</td><td>Number of messages stored in Early Queue since statistical reset, see also Chapter 4.6.8.</td></tr>
<tr><td>Bad messages</td><td>Number of rejected messages since statistics reset.</td></tr>
<tr><td>Frame no.</td><td>Sent counter (rotation).</td></tr>
<tr><td>Channel state</td><td>Current state of Channel 1.
The channel state is the current state of Channel 1 to the time (Seq. no X-1) when a message with Seq. no. X is received.
<table>
<tr><th>Status</th><th>Description</th></tr>
<tr><td>0</td><td>No information on the state of Channel 1.</td></tr>
<tr><td>1</td><td>Channel 1 OK.</td></tr>
<tr><td>2</td><td>The last message was wrong, the current one is OK.</td></tr>
<tr><td>3</td><td>Fault on Channel 1.</td></tr>
</table></td></tr>
<tr><td>Layout Version</td><td>Signature of the data layout used within communication.</td></tr>
<tr><td>Last channel latency</td><td rowspan="9">The channel latency specifies the delay between two redundant transmission paths to the reception time of messages with identical Seq. no.
A statistic is kept specifying the average, minimum, maximum and last latency.
If the minimum value is greater that the maximum value, the statistic values are invalid.
The last channel latency and the average channel latency are then 0.</td></tr>
<tr><td>Last latency of the red. channel</td></tr>
<tr><td>Max. channel latency</td></tr>
<tr><td>Max. latency of the red. channel</td></tr>
<tr><td>Min. channel latency</td></tr>
<tr><td>Min. latency of the red. channel</td></tr>
<tr><td>Average channel latency</td></tr>
</table>

<table>
<tr><td>Average latency of the red. channel</td><td></td></tr>
<tr><td>Monotony</td><td>Count for user data sending (rotation).</td></tr>
<tr><td>New Layout Version</td><td>Signature of the new data layout.</td></tr>
<tr><td>Quality of Channel 1</td><td>State of the main transmission path.
<table>
<tr><th>Bit no.</th><th>Bit = 0</th><th>Bit = 1</th></tr>
<tr><td>0</td><td>Transmission path not activated</td><td>Transmission path activated</td></tr>
<tr><td>1</td><td>Transmission path not used</td><td>Transmission path actively used</td></tr>
<tr><td>2</td><td>Transmission path not connected</td><td>Transmission path connected</td></tr>
<tr><td>3</td><td>-</td><td>Transmission path first provides message</td></tr>
<tr><td>4 - 7</td><td>reserved</td><td>reserved</td></tr>
</table></td></tr>
<tr><td>Quality of Channel 2</td><td>State of the redundant transmission path, see state of Channel 1 (main transmission path).</td></tr>
<tr><td>Receive Timeout</td><td>Time in milliseconds (ms) of PES1 within which PES2 must receive a valid response.
See also Chapter 4.6.2.</td></tr>
<tr><td>Response Time</td><td>Time in milliseconds (ms) until the acknowledgment of a message is received by the sender, see also Chapter 4.6.3.</td></tr>
<tr><td>Reset safeethernet statistics</td><td>Reset the statistical values for the communication connection in the user program. To do this, set the Reset Statistics parameter from FALSE to TRUE.</td></tr>
<tr><td>Transmission Control Ch1</td><td>Transmission control of Channel 1
<table>
<tr><th>Bit 0</th><th>Function</th></tr>
<tr><td>FALSE</td><td>Transmission path activated</td></tr>
<tr><td>TRUE</td><td>Transmission path locked</td></tr>
</table>
<table>
<tr><th>Bit 1</th><th>Function</th></tr>
<tr><td>FALSE</td><td>Transmission path activated for tests</td></tr>
<tr><td>TRUE</td><td>Transmission path locked</td></tr>
</table>
Bits 2...7 reserved.</td></tr>
<tr><td>Transmission Control Ch2</td><td>Transmission control of Channel 2,
see transmission control of Channel 1.</td></tr>
</table>

<table>
<tr><td>Connection Control</td><td>Use this system variable to control the safeethernet connection from within the user program.
<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>Autoconnect (0x0000)</td><td>Default value:
After a safeethernet communication loss, the controller attempts to re-establish the connection in the following CPU cycle.</td></tr>
<tr><td>Toggle Mode 0 (0x0100)
Toggle Mode 1 (0x0101)</td><td>After a communication loss, the connection can be re-established performing a program-controlled change of the toggle mode.
▪ TOGGLE MODE_0 (0x100) set:
Set to TOGGLE MODE 1 (0x101) to re-establish the connection.
▪ TOGGLE MODE_1 (0x101) set:
Set to TOGGLE MODE 0 (0x100) to re-establish the connection.</td></tr>
<tr><td>Disabled (0x8000)</td><td>safeethernet communication is disabled.</td></tr>
</table></td></tr>
<tr><td>Connection State</td><td>Connection State is used to evaluate the communication status between two controllers from within the user program.
<table>
<tr><th>Status/Value</th><th>Description</th></tr>
<tr><td>Closed (0)</td><td>The connection is closed and no attempt is made to open it.</td></tr>
<tr><td>Try_open (1)</td><td>An attempt is made to open the connection, but it is still closed. This state applies for both the active and the passive sides.</td></tr>
<tr><td>Connected (2)</td><td>The connection is established and functioning (active time monitoring and data exchange).</td></tr>
</table></td></tr>
<tr><td>Repeats</td><td>Number of reconnections since statistics reset [UDINT].</td></tr>
<tr><td>Timestamp for the last fault on the red. channel [ms]</td><td>Millisecond fraction of the timestamp (current system time)</td></tr>
<tr><td>Timestamp for the last fault on the red. channel [s]</td><td>Second fraction of the timestamp (current system time)</td></tr>
<tr><td>Timestamp for the last fault [ms]</td><td>Millisecond fraction of the timestamp (current system time)</td></tr>
<tr><td>Timestamp for the last fault [s]</td><td>Second fraction of the timestamp (current system time)</td></tr>
</table>

<table>
<tr><td>State of the red. channel</td><td>Current state of Channel 2.
The channel state is the current state of Channel 2 to the time (Seq. no X-1) when a message with Seq. no. X is received.
<table>
<tr><th>Status</th><th>Description</th></tr>
<tr><td>0</td><td>No information on the state of Channel 2.</td></tr>
<tr><td>1</td><td>Channel 2 OK.</td></tr>
<tr><td>2</td><td>The last message was wrong, the current one is OK.</td></tr>
<tr><td>3</td><td>Fault on Channel 2.</td></tr>
</table></td></tr>
</table>

Table 26: System Variables Tab in the safe**ethernet** Editor

4.5 Possible safeethernet Connections

A safe**ethernet** connection between two HIMax controllers can be configured as mono or redundant.

The Ethernet interfaces available for a safe**ethernet** connection are always displayed related to the resource (local) for which the safe**ethernet** Editor was opened.

All Ethernet interfaces available for a controller are showed in the drop-down menu for the **IF CH...** parameter.

Element	Description
IF CH1 (local)	Ethernet interface (channel 1) of the resource
IF CH2 (local)	Ethernet interface (channel 2) of the resource
IF CH1 (target)	Ethernet interface (channel 1) of the link partner
IF CH2 (target)	Ethernet interface (channel 2) of the link partner

Table 27: Available Ethernet Interfaces

4.5.1 Mono safeethernet Connection (Channel 1)

For a mono connection, the Ethernet interfaces *IF Ch1 (local)* and *IF Ch1 (target)* must be set in the local resource.

4.5.2 Redundant safeethernet Connection (Channel 1 and Channel 2)

Redundant safe**ethernet** transmission paths between two HIMax/HIMax controllers are possible.

i When doing so, the bandwidth and the delay on the two transmission paths must be nearly identical.

For a redundant connection, the following Ethernet interfaces can be used:

- The Ethernet interface *IF Ch1 (local)* and *IF Ch1 (target)* for channel 1.
- The Ethernet interfaces *IF Ch2 (local)* and *IF Ch2 (target)* for channel 2.
- For a redundant connection via Channel 1 and Channel 2 using only one Ethernet interface, select the same Ethernet interface *IF CH1 (local)* Channel 1 and *IF CH2 (local)* Channel 2 in the safe**ethernet** Editor.

i The reciprocal communication path is automatically added in the safe**ethernet** Editor of the target resource.

4.5.3 Permitted Combinations

The following table lists the possible combinations for redundant safe**ethernet** connections.

Channel 1 IF Ch1 (local) / IF Ch1 (target)	Channel 2 IF Ch2 (local) / IF Ch2 (target)
CPU1/CPU1	CPU2/CPU2
CPU1/CPU1	CPU1/CPU1
COM1/COM1	COM2/COM2
CPU1/COM1	CPU2/COM2
CPU1/COM2	CPU2/COM1
CPU1/CPU1	COM1/COM1

Table 28: Combinations for safe**ethernet** Connections

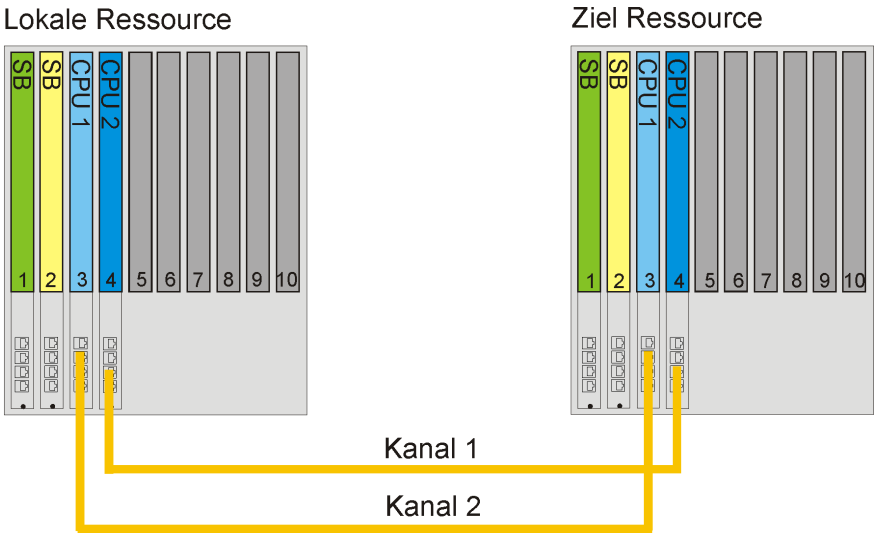


Figure 7: Redundant Connection between Two HIMax Controllers

To achieve a constant data rate despite lost data packets, the following redundant connection can be used (via a line).

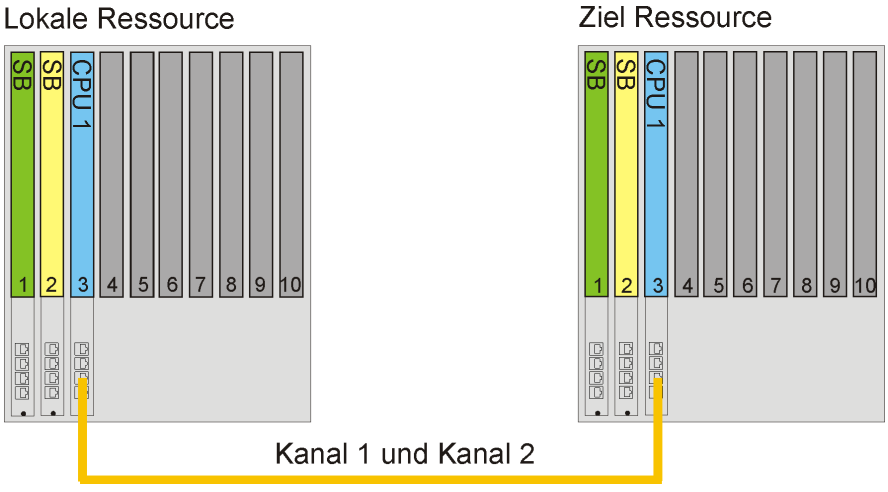


Figure 8: Redundant Connection of Two HIMax Controllers using a Line

4.6 safeethernet Parameters

The safety-related communication is configured in the safe**ethernet** Editor. To do this, the parameters described in this chapter must be set.

4.6.1 Maximum Cycle Time (Minimum Watchdog Time) of the HIMax Controller

To determine the maximum cycle time for a HIMax controller (minimum watchdog time), HIMA recommends proceeding as follows when all the processor modules of the system are inserted.

1. Set the watchdog time high for testing.
2. Use the system under the maximum load: In the process, all communication connections must be operating both via safeethernet and standard protocols. Frequently read the cycle time in the Control Panel and note the variations of the cycle time..
3. In succession, remove and reinsert every processor module in the base plate. Prior to removing one processor module, wait that the processor module that has just been inserted is synchronized.

i When a processor module is inserted in the base plate, it automatically synchronizes itself with the configuration of the existing processor modules. The time required for the synchronization process extends the controller cycle up to the maximum cycle time.

The synchronization time increases with the number of processor modules that have already been synchronized.

For more information on how to insert and remove a processor module, refer to the X-CPU 01 manual HI 801 009.

4. In the diagnostic history, read the synchronization time from n to n+1 processor modules in every synchronization process and not it down.
5. Repeat these steps for the second communication partner (i.e., the second HIMax controller). The longest synchronization time is used to determine the watchdog time.

i Note down the synchronization times of both HIMax controllers!

6. Calculate the minimum watchdog time from the longest synchronization time + 12 ms spare + spare for the noted variations of the cycle time.

A suitable value for the maximum cycle time (minimum watchdog time) has been thus determined for the following calculations.

TIP Perform the calculations specified in step 6 for both HIMax controllers and use the corresponding synchronization time value previously noted down.

The maximum cycle times (minimum watchdog time) calculated as described above can be used as watchdog time in the corresponding resource, see Safety Manual HI 801 003.

4.6.2 Receive Timeout

ReceiveTMO is the monitoring time in milliseconds (ms) within which a correct response from the communication partner must be received.

If a correct response is not received from the communication partner within *ReceiveTMO*, safety-related communication is terminated. The input variables of this safe**ethernet** connection react in accordance with the preset parameter *Freeze Data on Lost Connection [ms]*.

For safety-related functions implemented via safe**ethernet**, only the **Use Initial Data** setting may be used.

Since *ReceiveTMO* is a safety-relevant component of the Worst Case Reaction Time T_R (see Chapter **4.7.1** et seqq.), its value must be determined as described below and entered in the safe**ethernet** Editor.

ReceiveTMO ≥ 4*delay + 5*max. cycle time

Condition: The Communication Time Slice must be sufficiently high to allow all the safeethernet connections to be processed within one CPU cycle.

Delay:	Delay on the transmission path, e.g., due to switch or satellite.
Max. Cycle Time	Maximum cycle time of both controllers.

i A wanted fault tolerance of communication can be achieved by increasing *ReceiveTMO,* provided that this is permissible in terms of time for the application process.

4.6.3 Response Time

ResponseTime is the time in milliseconds (ms) that elapses until the sender of the message receives acknowledgement from the recipient.

When configuring using a safe**ethernet** profile, a *Response Time* parameter must be set based on the physical conditions of the transmission path.

The preset *ResponseTime* affects the configuration of all the safe**ethernet** connection parameters and is calculated as follows:

ResponseTime ≤ ReceiveTMO / n

n = 2, 3, 4, 5, 6, 7, 8.....

The ratio between *ReceiveTMO* and *ResponseTime* influences the capability to tolerate faults, e.g., when packets are lost (resending lost data packets) or delays occur on the transmission path.

In networks where packets can be lost, the following condition must be given:

min. Response Time ≤ ReceiveTMO / 2 ≥ 2*Delay + 2.5*max. Cycle Time

If this condition is met, the loss of at least one data packet can be intercepted without interrupting the safe**ethernet** connection.

i If this condition is **not met**, the availability of a safeethernet connection can only be ensured in a collision and fault-free network. However, this is not a safety problem for the processor module!

i Make sure that the communication system complies with the configured response time!

If this conditions cannot always be ensured, a corresponding connection system variable for monitoring the response time is available. If the measured response time is not seldom exceeded for over the half P2P ReceiveTMO, the configured response time must be increased.

The receive timeout must be adjusted according to the new value configured for response time.

4.6.4 Sync/Async

Sync Currently not supported.

Async is the default setting.
With the Async setting, the safe**ethernet** protocol instance receives during the CPU input phase and sends in accordance with its send rules during the CPU output phase.

4.6.5 ResendTMO

ResendTMO cannot be set manually, but it is calculated based on the profile and Response Time.

Monitoring time in milliseconds (ms) of PES1 within which PES2 must have acknowledged the reception of a data packet; otherwise the data packet is resent.

Rule: *ResendTMO* ≤ *Receive-Timeout*

If the *ResendTMO* set for the #communication partners differ from one another, the active protocol partner (the lowest SRS) determines the value for the *ResendTMO* of the protocol connection.

4.6.6 Acknowledge Timeout

AckTMO cannot be set manually, but it is calculated based on the profile and Response Time.

AckTMO is the time period within which the CPU must acknowledge the reception of a data packet.

In a rapid network, *AckTMO* is zero, i.e., the reception of a data packet is immediately acknowledged. In a slow network (e.g., a telephone modem line), *AckTMO* is greater than zero. In this case, the system attempts to transmit the acknowledgment message together with the process data to reduce the network load by avoiding addressing and security blocks.

Rules:

- *AckTMO* must be ≤ *Receive Timeout*
- *AckTMO* must be ≤ *ResendTMO* if *ProdRate* is > *ResendTMO*.

4.6.7 Production Rate

ProdRate cannot be set manually, but it is calculated based on the profile and Response Time.

Minimum time interval in milliseconds (ms) between two data packets.

The *ProdRate* is used to limit the amount of data packets such that a (slow) communication channel will not be overloaded. This ensures a uniform load of the transmission medium and prevents the receiver from receiving obsolete data.

Rules:

- *ProdRate* ≤ *Receive Timeout*
- *ProdRate* ≤ *Resend Timeout*, if *Acknowledge Timeout* > *Resend Timeout*

i A zero production rate means that data packets can be transmitted in each user program cycle.

4.6.8 Queue

Queue cannot be set manually, but it is calculated based on the profile and Response Time.

Queue is the number of data packets that can be sent without waiting for their acknowledgement.

The value depends on the network transfer capacity and potential network delays.

All safeethernet connections share the message queue available in the CPU.

4.7 Worst Case Reaction Time for safeethernet

In the following examples, the formulas for calculating the worst case reaction time only apply for a connection with HIMatrix controllers if the parameter Safety Time = 2 * Watchdog Time is set. These formulas always apply to HIMax controllers.

i The allowed worst case reaction time depends on the process and must be agreed upon together with the test authority responsible for the final inspection.

Terms

ReceiveTMO:	Monitoring time of PES 1 within which a correct response from PES 2 must be received. Otherwise, safety-related communication is terminated after the time has expired.
Production Rate:	Minimum interval between two data transmissions.
Watchdog Time:	Maximum duration permitted for a controller's RUN cycle. The duration of the RUN cycle depends on the complexity of the user program and the number of safe**ethernet** connections. The watchdog time (WDT) must be entered in the resource properties.
Worst Case Reaction Time	The worst case reaction time is the time between a change in a physical input signal (in) of PES 1 and a reaction on the corresponding output (out) of PES 2.
Delay:	Delay of a transmission path e.g., with a modem or satellite connection. For direct connections, an initial delay of 2 ms can be assumed. The responsible network administrator can measure the actual delay on a transmission path.

To the calculations of the maximum reaction times specified below, the following conditions apply:

- The signals transmitted over safe**ethernet** must be processed in the corresponding controllers within one CPU cycle.
- Further, the reaction time of the sensors and actuators must be added.

The calculations also apply to signals in the opposite direction.

4.7.1 Calculating the Worst Case Reaction Time of Two HIMax Controllers

The worst case reaction time T_R is the time between a change on the sensor input signal (in) of PES 1 and a reaction on the corresponding output (out) of PES 2. It is calculated as follows

Figure 9: safe**ethernet** Connection of Two HIMax Controllers

Reaction Time when two HIMax controllers are interconnected

$T_R = t_1 + t_2 + t_3$

T_R Worst Case Reaction Time

t_1 Safety Time of PES 1

t_2 *ReceiveTMO*

t_3 Safety Time of PES 2

4.7.2 Calculating the Worst Case Reaction Time in Connection with One HIMatrix PES

Reaction time T_R between a change on the sensor input signal (in) of HIMax PES 1 and a reaction on the corresponding output (out) of HIMatrix PES 2. It is calculated as follows:

Figure 10: safe**ethernet** Connection between One HIMax and One HIMatrix Controller

Reaction time when one HIMax controller is connected to one HIMatrix controller:

$T_R = t_1 + t_2 + t_3$

T_R Worst Case Reaction Time

t_1 Safety Time of HIMax PES 1

t_2 *ReceiveTMO*

t_3 2 * Watchdog Time of HIMatrix PES 2

4.7.3 Calculating the Worst Case Reaction Time with two HIMatrix Controllers or RIOs

The worst case reaction time T_R is the time between a change on the sensor input signal (in) of the first HIMatrix PES 1 or RIO (e.g., F3 DIO 20/8) and a reaction on the corresponding output (out) of the second HIMatrix PES 2 or RIO. It is calculated as follows

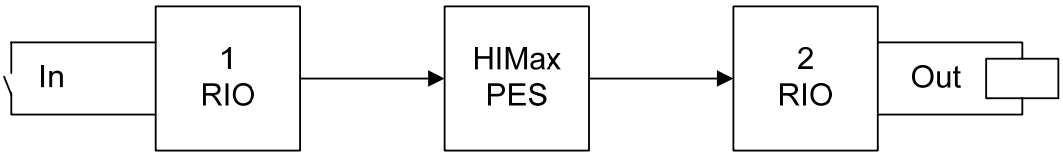


Figure 11: safe**ethernet** Connection in Connection with RIOs

Response Time with RIOs

$T_R = t_1 + t_2 + t_3 + t_4 + t_5$

T_R Worst Case Reaction Time

t_1 2 * watchdog time of the 1st RIO

t_2 *ReceiveTMO1*

t_3 2 * Watchdog Time of HIMax PES

t_4 *ReceiveTMO2*

t_5 2 * watchdog time of the 2nd RIO

i The two RIOs can also be identical. The time values still apply if a HIMatrix PES is used instead of a RIO.

4.7.4 Calculating the Worst Case Reaction Time with Two HIMax and One HIMatrix PES

Worst case reaction time T_R between a change on the sensor input signal (in) of the first HIMax PES and a reaction on the corresponding output (out) of the second HIMax PES. It is calculated as follows

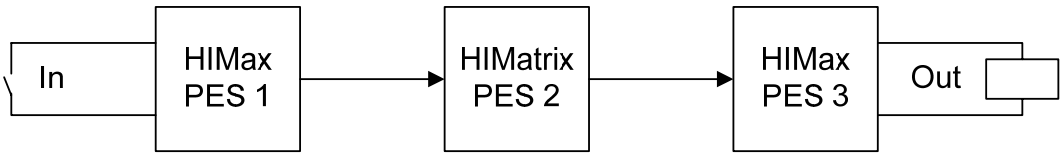


Figure 12: safe**ethernet** Connection between Two HIMax and One HIMatrix PES

Reaction time when two HIMax controllers are connected to one HIMatrix controller:

$T_R = t_1 + t_2 + t_3 + t_4 + t_5$

T_R Worst Case Reaction Time

t_1 Safety Time of HIMax PES 1

t_2 *ReceiveTMO1*

t_3 2 * Watchdog Time of HIMatrix PES 2

t_4 *ReceiveTMO2*

t_5 Safety Time of HIMax PES 3

i HIMax PES 1 and HIMax PES 3 can also be identical.
HIMatrix PES 2 can also be a HIMax PES.

4.7.5 safeethernet Profile

safe**ethernet** profiles are combinations of parameters compatible with one another that are automatically set when one of the safe**ethernet** profiles is selected.
When configuring, only the Receive Timeout and the expected Response Time parameters must be individually set.
A safe**ethernet** profile is used to optimize the data throughput within a network taking the physical conditions into account.

To ensure that the optimization is effective the following conditions must be met:

- the Communication Time Slice must be sufficiently high to allow all the safeethernet connections to be processed within one CPU cycle.
- if average CPU cycle time < response time.
- if average CPU cycle time < ProdRate or ProdRate = 0.

NOTE

Disturbance of the safe**ethernet communication up to communication loss!**

Unsuitable combinations of CPU cycle, communication time slice, response time and ProdRate are not rejected during code generation and download/reload, but can cause communication disturbances.

In the Control Panel, verify the *Bad Messages* and *Resends* values for both controllers.

Six safe**ethernet** profiles are available. Select the safe**ethernet** profile the most suitable for the transmission path.

For safety-related process data communication, only the profiles "Fast&Noisy", "Medium&Noisy" and "Slow&Noisy" may be used.

Fast & Cleanroom	Not suitable foor safety-related process data communication!
Fast & Noisy	
Medium & Cleanroom	Not suitable foor safety-related process data communication!
Medium & Noisy	
Slow & Cleanroom	Not suitable foor safety-related process data communication!
Slow & Noisy	

4.7.6 Profile I (Fast & Cleanroom)

NOTE

Not suitable foor safety-related process data communication!
For safety-related process data communication, only the profiles "Fast&Noisy", "Medium&Noisy" and "Slow&Noisy" may be used.

Use

The *Fast & Cleanroom* profile is suitable applications in ideal environments such as laboratories!

- For the fastest data throughput
- For applications requiring fast data transmission
- For application requiring a worst case reaction time as low as possible

Network requirements:

- Fast: 100 Gbit/100 Mbit technology, 1 Gbit technology
- Clean: trouble-free network. Avoid data loss due to network overload, external influences or network manipulation.
- LAN switches are necessary!

Communication path characteristics:

- Minimum delays
- Expected ResponseTime ≤ ReceiveTMO
  (otherwise ERROR during configuration)

4.7.7 Profile II (Fast & Noisy)

Use

The *Fast & Noisy* profile is the SILworX default profile for communicating via safe**ethernet**.

- For fast data throughput
- For applications requiring fast data transmission
- For application requiring a worst case reaction time as low as possible

Network requirements:

- Fast: 100 Gbit/100 Mbit technology, 1 Gbit technology
- Noisy: Non-trouble-free network.
  Low probability of data packet loss
  time for ≥ 1 resends
- LAN switches are necessary!

Communication path characteristics:

- Minimum delays
- Expected ResponseTime ≤ ReceiveTMO / 2
  (otherwise ERROR during configuration)

4.7.8 Profile III (Medium & Cleanroom)

NOTE

Not suitable foor safety-related process data communication!
For safety-related process data communication, only the profiles "Fast&Noisy", "Medium&Noisy" and "Slow&Noisy" may be used.

Use

The *Medium & Cleanroom* profile is only suitable in a trouble-free network for applications that require a moderate fast data transmission.

- For medium data throughput
- Suitable for Virtual Private Networks (VPN) in which data is slow but fault-free as it is exchanged via intermediate safety devices (firewalls, encryption).
- Suitable for applications in which the worst case reaction time is not a critical factor

Network requirements:

- Medium: 10 Mbit (10 Base T), 100 Mbit (100 Base TX), 1 Gbit technology
- LAN switches are necessary!
- Clean: trouble-free network. Avoid data loss due to network overload, external influences or network manipulation.
  Time for ≥ 0 resends

Communication path characteristics:

- Moderate delays
- Expected ResponseTime ≤ ReceiveTMO
  (otherwise ERROR during configuration)

4.7.9 Profile IV (Medium & Noisy)

Use

The *Medium & Noisy* profile is suitable for applications that require moderate fast data transmission

- For medium data throughput
- For applications requiring moderate fast data transmission
- Suitable for applications in which the worst case reaction time is not a critical factor

Network requirements:

- Medium: 10 Mbit (10 Base T), 100 Mbit (100 Base TX), 1 Gbit technology
- LAN switches are necessary!
- Noisy: Non-trouble-free network.
  Low probability of data packet loss
  time for ≥ 1 resends

Communication path characteristics:

- Moderate delays
- Expected ResponseTime ≤ ReceiveTMO / 2
  (otherwise ERROR during configuration)

4.7.10 Profile V (Slow & Cleanroom)

NOTE

Not suitable foor safety-related process data communication!
For safety-related process data communication, only the profiles "Fast&Noisy", "Medium&Noisy" and "Slow&Noisy" may be used.

Use

The *Slow & Cleanroom* profile is suitable for applications in a trouble-free network that require a slow data transmission.

- For slow data throughput
- For applications that only require a slow data transmission to controllers (potentially located far away) or if the communication path conditions cannot be anticipated.

Network requirements:

- Slow: Data transfer via ISDN, dedicated line or radio relay.
- Clean: trouble-free network. Avoid data loss due to network overload, external influences or network manipulation.
  Time for ≥ 0 resends

Communication path characteristics:

- Moderate delays
- Expected ResponseTime = ReceiveTMO
  (otherwise ERROR during configuration)

4.7.11 Profile VI (Slow & Noisy)

Use

The Slow & Noisy profile is suitable for applications that only require a slow data transmission to controllers (potentially located far away).

- For slow data throughput
- For applications; generally for data transfer via bad telephone lines or disturbed radio relays.

Network requirements:

- Slow: Data transfer via telephone, satellite, radio etc.
- Noisy: Non-trouble-free network.
  Low probability of data packet loss
  time for ≥ 1 resends

Communication path characteristics:

- Moderate to important delays
- Expected ResponseTime ≤ ReceiveTMO / 2
  (otherwise ERROR during configuration)

4.8 Cross-Project Communication

Cross-project communication is used to ensure that resources located in different projects can exchange process variables with one another.

The communication between two projects is occurs via safe**ethernet** and is configured in the safe**ethernet** Editor.

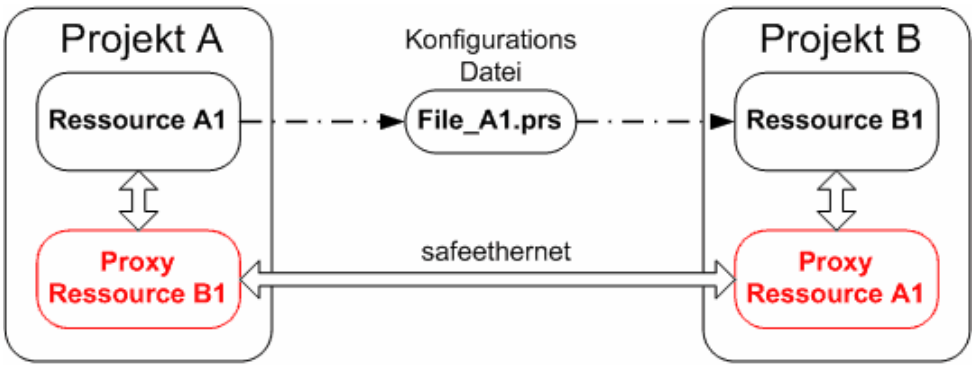


Figure 13: safe**ethernet** Connection between Resource A1 in Project A and Resource B1 in Project B

The project in which the safe**ethernet** connection is configured and the configuration file is created is referred to as 'local project'.

The project to which the configuration file is imported is referred to as 'target project'.

The local and the target projects are equal communication partners when they exchange data.

A proxy resource serves as placeholder for the corresponding resource from an external project and is used for importing and exporting the safe**ethernet** connections.

Proxy resource B1 in *Project A* is the placeholder for *Resource B1* from *Project B*.

Proxy Resource A1 in *Project B* is the placeholder for *Resource A1* from *Project A*.

In the local project (in the example: *Project A*), the proxy resource (in the example: *Proxy Resource B1*) must be created and configured manually. Once the configuration is completed, the configuration file (in the example: *File_A1.prs*) must be imported to the target project (in the example: Resource B1).

The *File_A1.prs* configuration file contains the entire description of *Resource A1* for the safe**ethernet** connection with *Resource B1. Proxy Resource A1* is automatically created in *Project B* after importing the *File_A1.prs* configuration file to *Resource B1.*

4.8.1 Variants for Cross-Project Communication

In both of the following variants, Project A and Project B communicate with one another via safe**ethernet**.

The local project is Project A in the first variant and Project B in the second variant. The user may decide in which project the configuration should be created.

Both options require the same amount of work and result in the same configuration.

Local Project A

In the local Project A, the user configures communication with target Project B and creates the configuration files. This has the advantage that the user must only manually create Proxy Resource B1 in the local project.

Local project -> target project

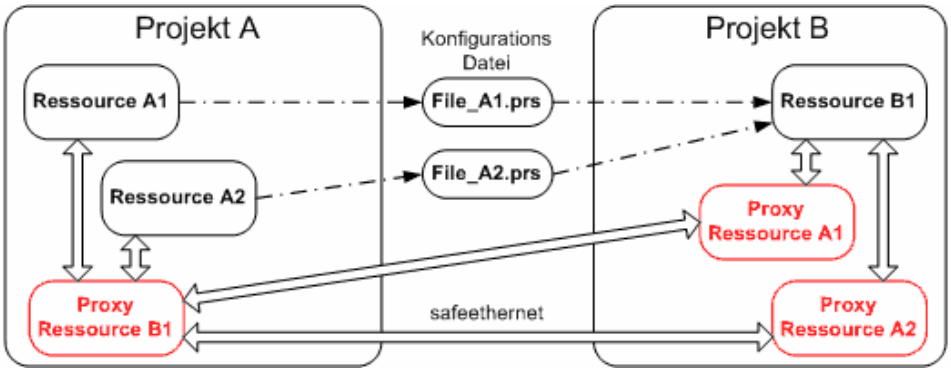


Figure 14: Variant: Project A as Local Project

Local Project B

In the local Project B, the user configures communication with target Project A and creates the configuration files. This has the disadvantage that the user must manually create both Proxy Resource A1 and Proxy Resource A2 in the local project B.

Local project -> target project

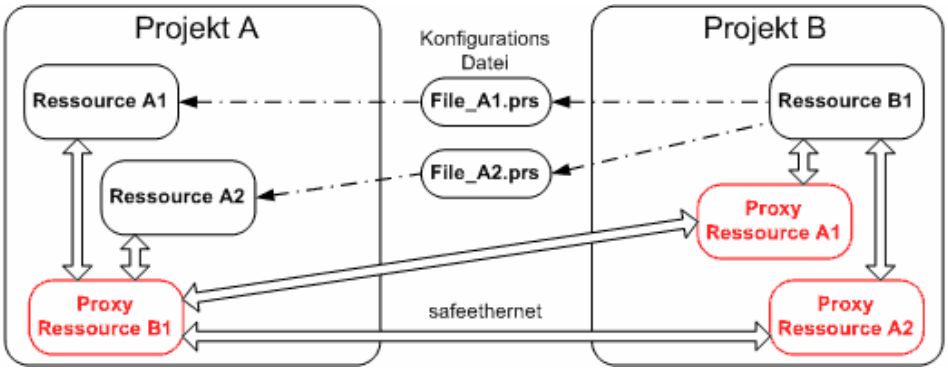


Figure 15: Variant: Project B as Local Project

4.9 Cross-Project Communication between SILworX and ELOP II Factory

This example shows how to configure a safe**ethernet** connection between HIMax and HIMatrix.

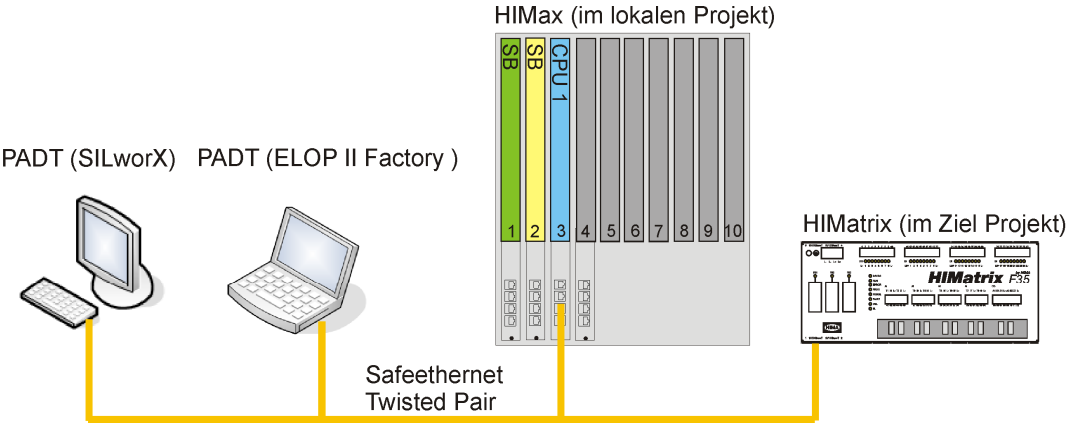


Figure 16: Configuring Communication between SILworX and ELOP II Factory

Open the resource in the target project (HIMatrix) that should serves as proxy resource in the local project (HIMax).

For this target resource, determine the following parameters:

- System ID
- Safety Time [ms]
- Watchdog Time [ms]
- IP Address

i The resource properties are safety-relevant and are subjected to restrictions. For more information, refer to the Safety Manual HI 801 003.

4.9.1 Configuring the HIMax in a SILworX Project

Creating the Proxy Resource

A proxy resource serves as placeholder for a resource in an external project and is used for importing and exporting the safe**ethernet** connections.

To create a proxy resource in the local project

1. Open the local project in which the proxy resource should be created.
2. In the structure tree, open **Configuration.**
3. Right-click **Configuration**, and then click **New, Proxy Resource ELOP II Factory**.
   ☑ A new proxy resource is created.

To configure a proxy resource in the local project

1. Right-click the proxy resource, and then click **Properties**.
2. Enter a unique name in the **Name** field.
   Use the name of the resource in the target project for the proxy resource in the local project.
3. Enter the **System ID**, **Safety Time [ms]** and **Watchdog Time [ms]** previously calculated for this proxy resource.
4. Click **OK**. The remaining parameters can retain the default values.

To open the structure tree for the proxy resource

1. Right-click **Hardware**, and then click **Edit**, **HIMatrix Proxy**.
2. Click **OK** to confirm. The Hardware Editor for the proxy resource appears.

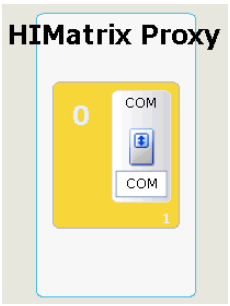


Figure 17: HIMatrix Proxy Resource

3. Double-click the **COM Module** and enter the **IP address** calculated for the proxy resource.
4. Click the **Save** button or **Save** on the **Window** menu, then click **Close**.
5. Repeat these steps for every further proxy resource contained in the local project.

Connecting the Local Resource to the Proxy Resource

In the safe**ethernet** Editor, create a safe**ethernet** connection between the locale and the proxy resource.

To open the safeethernet Editor of the local resource

1. In the structure tree, open **Configuration, Resource**.
2. Right-click safe**ethernet**, then select **Edit**. The new proxy resource is created in the Object Panel.

To create the safeethernet connection to the proxy resource

1. In the Object Panel, drag the **proxy resource** anywhere in the workspace of the safe**ethernet** Editor.
2. Select **Ethernet** interfaces on the local and proxy resource.

The following parameters determine the data throughput and the fault and collision tolerance of the safe**ethernet** connection.

7 Select the **Network Profile** for the safe**ethernet** connection (e.g., Fast&Noisy).
8. Calculate and enter **Receive Timeout** and **Response Time**.

Example of parameter values for a safe**ethernet** connection to a Proxy resource:

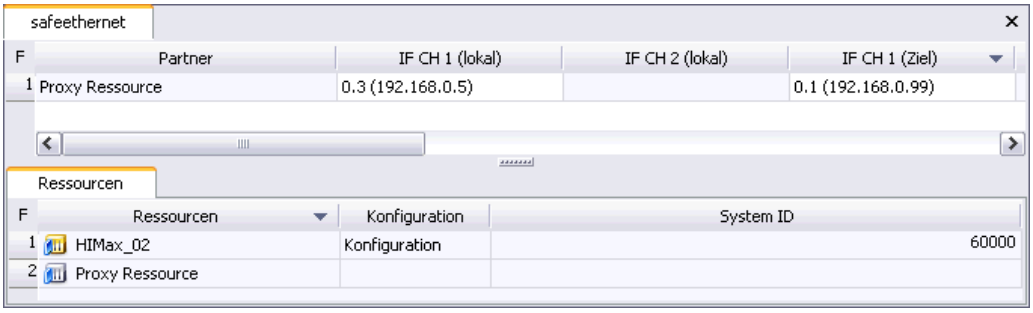


Figure 18: Parameter for a safe**ethernet** Connection to a Proxy Resource

Connecting Process Variables

Connect the process variables in the Detail View of the safe**ethernet** connection.

To open the Detail View of a safeethernet connection

Requirement: The safe**ethernet** Editor of the local resource must be opened.

1. Right-click the **Proxy Resource** line and open Proxy Resource.
2. Select **Detail View** on the context menu to open the Detail View of the safe**ethernet** connection..
3. Select the **Resource<->Proxy Resource** tab.

To add safeethernet send variables

Send variables are sent from the local to the proxy resource.

1. Select the **Resource->Proxy Resource** area.
2. In the Object Panel, select a **Global Variable** and drag it onto the **Resource (target)->Resource (local)** column.
3. Repeat these steps for every further safe**ethernet** send variables.

To add safeethernet receive variables

Receive variables are received by the local resource.

1. Select the **Resource<-Proxy Resource** area.
2. In the Object Panel, select a **Global Variable** and drag it onto the **Resource (target)<-Resource (local)** column.
3. Repeat these steps for every further safe**ethernet** receive variables.

Exporting the Configuration File from SILworX

The safe**ethernet** connection configured in SILworX must be exported as configuration file with the extension ***.prs**. This configuration file can be imported to ELOP II Factory to establish the safe**ethernet** connection for the HIMatrix controller.

To export a safeethernet connection

1. Click **Proxy Resource** in the safe**ethernet** Editor and open the context menu.
2. Click **Export Connection with Proxy Resource**:
   A standard dialog box for saving a file appears.
3. Enter a file name for the configuration file and save it with the extension ***.prs**.
4. Close the local project.

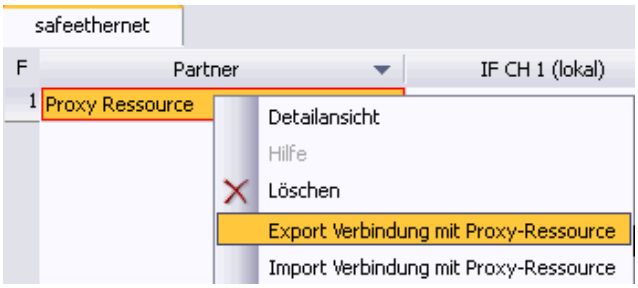


Figure 19: safe**ethernet** Connection Export

To verify the safeethernet connection

1. In the structure tree, open **Configuration, Resource, safeethernet.**
2. Click the **Verification** button on the **Action Bar**, and then click **OK** to confirm the action.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

i Recompile the configuration of the safe**ethernet** connection and the user program of the HIMax resource, and transfer them to the controllers. Only after this step, the new configurations can be used for communicating with HIMax system

4.9.2 Configuring a HIMatrix in an ELOP II Factory Project

To import the configuration file to the (HIMatrix) target project

1. Start ELOP II Factory.
2. Open the HIMatrix target project to which the configuration file should be imported.
3. In the structure tree, select the target resource and open the context menu.
4. Select **Import Connections**:
   A dialog box for importing a file with the extension ***.prs** opens.
5. Select the configuration file created in the local HIMax project, and click **OK**.
   ☑ Once the configuration file has been imported, the local HIMax resource is automatically created as proxy resource in the HIMatrix target project.

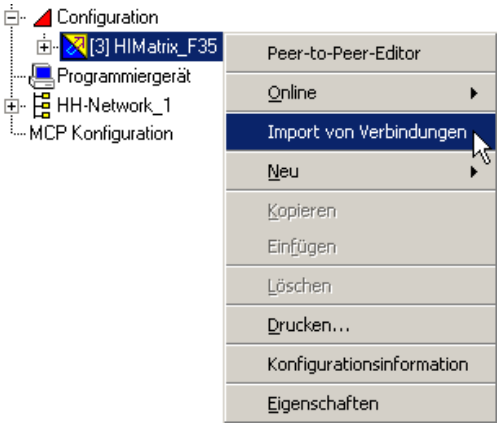


Figure 20: Importing Connections in ELOP II Factory

Assigning ELOP II Factory Process Signals

Connect process signals in the (HIMatrix) target resource.

Select **Signals, Editor** on the menu bar to open the **Signal Editor**.

To open the P2P Editor for the target resource in ELOP II Factory

1. In the structure tree, open **Configuration, Resource, P2P Editor**.
2. Enter the **HH Network** for this connection in the **P2P Editor**.
3. In the **P2P Editor**, click **Connect Process Signals**.

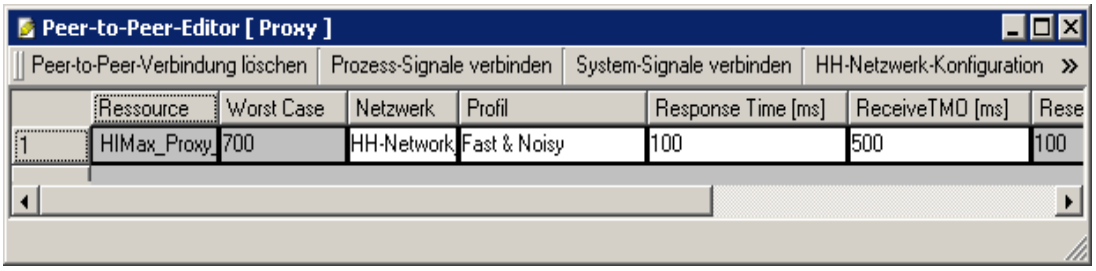


Figure 21: P2P Editor in ELOP II Factory

i Note that both communication partners must use the same profile and the same settings (automatically adopted while importing the configuration file).

To assign the P2P send signals

P2P send signals are sent from the HIMatrix resource to the HIMax resource.

1. Select the **HIMatrix Resource-> HIMax Proxy Resource** tab.
   The tab contains the imported P2P send signals..
2. In the Signal Editor, drag a **process signal** onto the send signal to be connected and located in the P2P Process Signals dialog box.
3. Repeat these steps for every further P2P send signal.

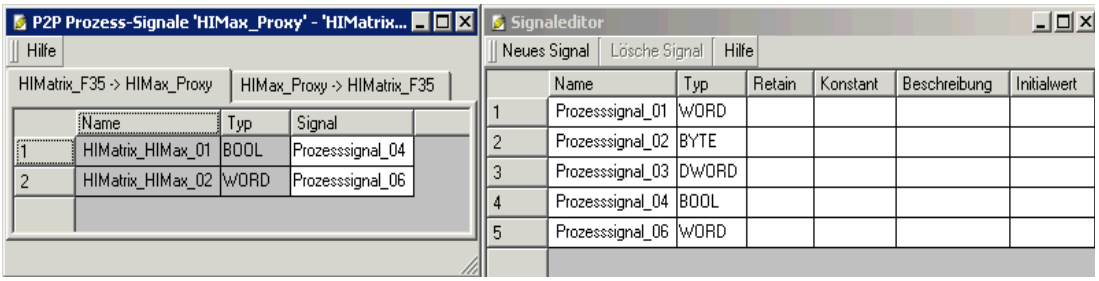


Figure 22: Assigning Send Signals in ELOP II Factory

To assign the P2P receive signals

P2P receive signals are received by the HIMatrix resource.

1. Select the **Resource<-Proxy Resource** tab.
   The tab contains the imported P2P receive signals.
2. In the Signal Editor, drag a **process signal** onto the receive signal to be connected and located in the P2P Process Signals dialog box.
3. Repeat these steps for every further P2P receive signal.

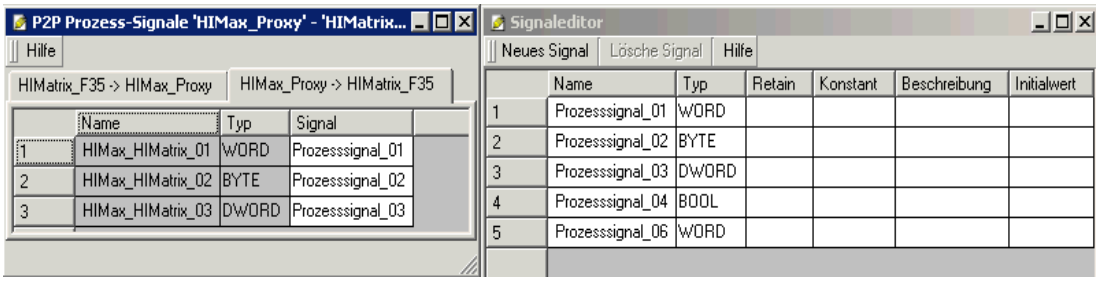


Figure 23: Assigning Receive Signals in ELOP II Factory

i For more information on how to connect process signals in ELOP II Factory, refer to the ELOP II Factory online help.

i Recompile the configuration of the P2P connection and the user program of the HIMatrix resource, and transfer them to the controllers. Only after this step, the P2P connection is active for the HIMatrix system.

4.10 Control Panel (safeethernet)

The Control Panel can be used to verify and control the safe**ethernet** connection settings. Details about the current status of the safe**ethernet** connection (e.g., cycle time, bus state, etc.) are also displayed.

To open Control Panel for monitoring the safeethernet connection

1. In the structure tree, click **Resource**.
2. Right click the resource, and then click **Online**.
3. In the **System Log-in** window, enter the access data to open the Control Panel for the resource.
4. In the structure tree associated with the Control Panel, select safe**ethernet**.

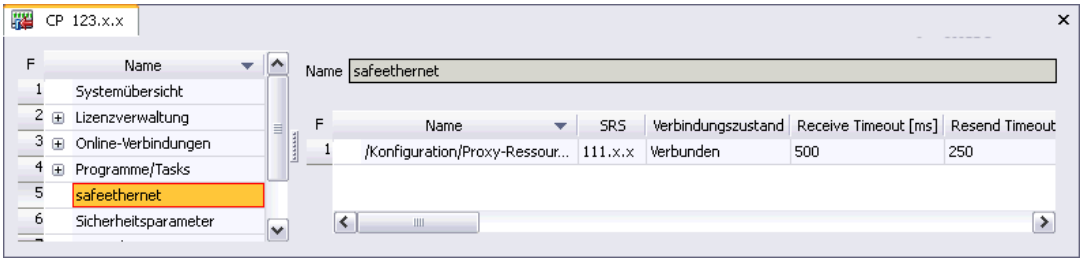


Figure 24: Control Panel for Connection Control

Reset statistical data

This function is used to reset the statistical data (cycle [min], cycle [max], etc.) to zero.

To reset the statistical data of the safeethernet connection

1. In the structure tree, select the safe**ethernet** connection.
2. Right-click the safe**ethernet** connection, and then select **Reset** safe**ethernet Statistics**.

4.10.1 View Box (safe**ethernet** Connection)

The view box displays the following values of the selected safe**ethernet** connection:

Element	Description
Name	Resource name of the communication partner
SRS	System.Rack.Slot
Connection State	State of the safe**ethernet** connection (See also Chapter 4.4)
Receive timeout [ms]	(See also Chapter 4.6.2)
Resend timeout [ms]	(See also Chapter 4.6.5)
Acknowledge timeout [ms]	(See also Chapter 4.6.6)
Min. RspT [ms]	Actual response time as minimum, maximum, last and average value. See also Chapter **4.6.3**.
Max. RspT [ms]	
Last RspT [ms]	
Av. RspT [ms]	
Bad Messages	Number of rejected messages since statistics reset.
Repeats	Number of reconnections since statistics reset [UDINT].
Number of Successful Connections	Number of successful connections since statistics reset.
Early Queue Usage	Number of messages stored in Early Queue since statistics reset. See also Chapter 4.6.8.
Frame No.	Current sent counter.
Ack.Frame No.	Current reception counter.
Monotony	Current count for user data sending.
Layout Version	Signature of the current communication send point.
New Layout Version	Signature of the new communication send point.
Connection Control	Connection control state. See also Chapter 4.4.
Transmission Control Ch1	Enable of transmission path Ch1. See also Chapter 4.4.
Transmission Control Ch2	Enable of transmission path Ch2. See also Chapter 4.4.
Quality of Channel 1	State of transmission path Ch1. See also Chapter 4.4.
Quality of Channel 2	State of transmission path Ch2. See also Chapter 4.4.
Late Received Redundant Messages	With redundant transmission paths. Number of late received messages since statistics reset.
Lost Redundant Messages	With redundant transmission paths. Number of messages received on one of the two transmission paths since statistics reset.
Protocol Version	0 and 1 = Precedent protocol version for HIMatrix < V7 2 = New protocol version for HIMax

Table 29: View Box of the safe**ethernet** connection

5 PROFINET IO

PROFINET IO is the transfer protocol provided by PNO Germany and is based on Ethernet technology.

With PROFINET IO, such as with PROFIBUS DP, the remote field devices are integrated in SILworX via a device description (GSDML file).

The HIMA PROFINET IO controller complies with Conformance Class A and supports non-real time (NRT) and real time (RT) communication with the PROFINET IO devices. In particular, real time communication is automatically used for time critical data exchange and non-real time communication for non time critical processes, such as acyclic read/write operations.

A redundant PROFINET IO connection can only be implemented by configuring a second PROFINET IO controller/device and adjusting it in the user program.

5.1 PROFINET IO Function Blocks

To acyclically exchange data, function blocks with the same functionality as with PROFIBUS DP are available in SILworX. The PROFINET IO function blocks are used to tailor the HIMA PROFINET IO controller and the corresponding PROFINET IO device to best meet the project requirements.

The following PROFINET IO function blocks are available:

Function block	Function description
MSTAT 6.9.1	Controlling the controller state using the user program
RALRM 6.9.2	Reading the alarm messages of the devices
RDREC 6.9.4	Reading the acyclic data records of the devices
SLACT 6.9.5	Controlling the device states using the user program
WRREC 6.9.6	Writing the acyclic data records of the devices

Table 30: Overview of PROFINET IO Function Blocks

The PROFINET IO function blocks are configured such as the PROFIBUS DP function blocks, see Chapter 6.9.

5.2 HIMA PROFINET IO Controller

This chapter describes the characteristics of the HIMA PROFINET IO controller and the menu functions and dialog boxes required to configure the HIMA PROFINET IO controller in SILworX.

5.3 System Requirements

Equipment and system requirements

Element	Description
Controller	HIMax with COM module
Processor module	The Ethernet interfaces on the processor module may not be used for PROFINET IO .
COM module	Ethernet 10/100BaseT.
Activation	Software activation code required, see Chapter 3.5.

Table 31: Equipment and System Requirements for the PROFINET IO Controller.

PROFINET IO Controller Properties

Element	Description
Safety-related	No
Transfer rate	100 Mbit/s full duplex
Transmission path	Ethernet interfaces on the COM module Ethernet interfaces in use can simultaneously be used for additional protocols.
Conformity class	The PROFINET IO controller meets the requirements for Conformance Class A.
Max. number of PROFINET IO controller	One PROFINET IO controller can be configured for each COM module.
Max. number of PROFINET IO devices application relations (ARs)	A PROFINET IO controller can establish an application relation (AR) with a maximum of 64 PROFINET IO devices.
Max. number of communication relations (CRs for each AR)	Max. 5 communication relations (CRs) for each AR und direction
Max. process data length of a CR	Output: max 1440 bytes Input: max. 1440 bytes (RT frame over UDP)
Data Priorization	Possible at device level using the *Reduction Rate* setting.
Interconnecting PROFINET and PROFIBUS	To do this, a PROFINET IO device with proxy functionality is required.

Table 32: PROFINET IO Controller Properties

5.4 PROFINET IO Example

This example illustrates how to connect the HIMA PROFINET IO controller to any kind of PROFINET IO devices.

5.4.1 Creating a HIMA PROFINET IO Controller in SILworX

To create a new HIMA PROFINET IO controller

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. Select **New**, **PROFINET IO Controller** on the context men for protocols to add a new PROFINET IO controller.
3. Select **Properties** on the context menu for PROFINET IO controller.
4. Click **COM Module**.

Configuring the PROFINET IO Device with SILworX

To create a HIMax PROFINET IO Device within the PROFINET IO controller

1. Select **New**, **PROFINET IO Device** on the context menu for the PROFINET IO controller.

To read the GSDML library file from an external data source (e.g., CD, USB stick, Internet):

1. On the structure tree, select **Configuration**, **Resource**, **Protocols**, **PROFINET IO Controller**, **GSDML Library**.
2. Select **Add GSDML File** on the context menu for the GSDML library and read the GSDML file specific to the PROFINET IO device.

---

i The GSDML library file usually contains several devices from one manufacturers.

To load the GSDML file for a new PROFINET IO device

1. On the structure tree, select **Configuration**, **Resource**, **Protocols**, **PROFINET IO Controller**, **PROFINET IO Device**.
2. Select **Properties** on the context menu and open the Parameter tab.

3. On the drop-down menu for **GSDML File,** select the GSDML library file specific to PROFINET IO device and close **Properties**.

To select the data access point (DAP) for the PROFINET IO device

1. On the structure tree, select **Configuration**, **Resource**, **Protocols**, **PROFINET IO Controller**, **PROFINET IO Device, DAP Module**.

2. Select **Select Device Access Point (DAP)** on the context menu and choose a suitable data record for the PROFINET IO device.

To verify the PROFINET IO configuration

1. In the structure tree, open **Configuration, Resource, Protocols, PROFINET IO Controller.**
2. Click the **Verification** button on Action Bar, and then click **OK** to confirm the action.

3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

Identifying the PROFINET IO Device within the Network

To find the PROFINET IO device within the Ethernet network

1. Log-in to the communication module containing the **PROFINET IO controller**.
2. On the structure tree corresponding to the Online View, select **PROFINET IO Controller, PROFINET IO Station**.
3. Select **Get PROFINET IO Network Stations**.
   ☑ A list appears specifying all the PROFINET devices in the network of the current PROFINET IO controller.

To configure the PROFINET IO device in the Online View:

1. To change the settings, right-click in the list the PROFINET IO device that should be configured.
2. Name the device using the **Name the PROFINET IO Device**, context menu function.
   ☑ Make sure that the PROFINET IO device name match the project. The name is case sensitive!
3. Set the IP address, subnet mask and the gateway using the **Network Settings** context menu function.

i In SILworX, the network settings for the PROFINET IO device must be configured in the PROFINET IO controller, or no communication is possible.

5.5 Menu Function in the PROFINET IO Controller

5.5.1 Properties

The **Properties** menu function on the context menu for the PROFINET IO controller opens the **Properties** dialog box.

Element	Description
Type	PROFINET IO Controller
Name	Any unique name for a PROFINET IO controller
Refresh Rate [ms]	Refresh rate in milliseconds at which the COM and CPU exchange protocol data. If the *Refresh Rate* is zero or lower than the cycle time for the controller, data is exchanged as fast as possible. Range of values: 4...(2^{31}-1). Default value: 0
Within one cycle	Activated: Transfer of all protocol data from the CPU to the COM within a CPU cycle. Deactivated: Transfer of all protocol data from the CPU to the COM, distributed over multiple CPU cycles, each with 1100 byte per data direction. This can also allow lowering the cycle time of the controller. Default value: Activated
Module	Selection of the COM module within which the protocol is processed.
Use Max CPU Load	Activated: Use CPU load limit from the field *Max. CPU Load [%]* Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%
RPC Port Server	Remote Procedure Call Port Range of values: 1024...65535 Default value: 49152 RPC port server and RPC port client must not be identical!
RPC Port Client	Remote Procedure Call Port Range of values: 1024...65535 Default value: 49153 RPC port server and RPC port client must not be identical!

Table 33: PROFINET IO Controller General Properties

5.6 Menu Functions for PROFINET IO Device (within the Controller)

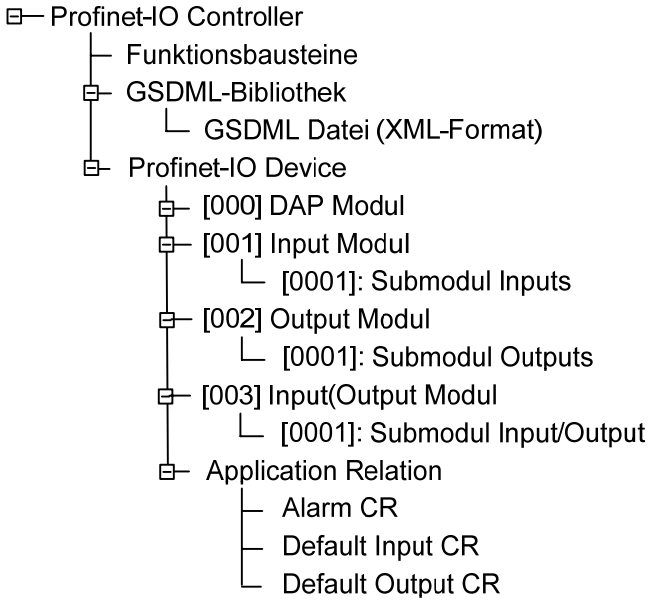


Figure 25: Structure Tree for the PROFINET IO Controller

5.6.1 Properties

The **Properties** menu function on the context menu for the PROFINET IO device opens the **Properties** dialog box. The dialog box contains the following tabs:

Tab Parameter

Element	Description
Name	Any unique name for the PROFINET IO device
Slot	Range of values: 0...125 Default value: 0
IP Address	Target IP address of the communication partner (with direct host routing) or network address (with subnet routing). Range of values: 0.0.0.0 ... 255.255.255.255 Default value: 0.0.0.0 Do not use IP addresses already in use, see Chapter 3.6.3.
Subnet Mask	Define the target address range for a routing entry. 255.255.255.255 (with direct host routing) or subnet mask of the addressed subnet. Range of values: 0.0.0.0 ... 255.255.255.255 Default value: 255.255.255.255
Gsdml File	GSDML stands for Generic Station Description Markup Language and refers to an XML-based description language. The GSDML file contains the PROFINET device master data

Table 34: Parameter Tab des PROFINET-IO Device

The tabs **Model** and **Features** are self-explanatory providing an additional description and specifying the equipment of the input/output PROFINET IO modules.

5.6.2 DAP Module (Device Access Point Module)

The DAP module (Device Access Point module) is used for connecting the bus, it always accompanies a PROFINET device and is located below it. The DAP module is a default directory and cannot be deleted.

The **Properties** function on the context menu for the DAP module opens the **Properties** dialog box. The dialog box contains the following tabs:

Tab Parameter

Element	Description
Name	Any unique name for the DAP module.
Slot	Not changeable Default value: 0

Table 35: Parameter Tab in the Properties Dialog Box for the DAP Module

The tabs **Model** and **Features** are self-explanatory and provide an additional description of the DAP module.

5.6.3 Input/Output PROFINET IO Modules

The PROFINET IO input modules are used to enter the HIMax PROFINET IO controller input variables that are sent by the PROFINET IO device.

The PROFINET IO output modules are used to enter the HIMax PROFINET IO controller output variables that are sent to the PROFINET IO device.

To create the required PROFINET IO Modules

1. In the structure tree, open **Configuration, Resource, Protocols, PROFINET IO Device.**
2. Select **New** on the context menu for the PROFINET IO device.
3. Select the modules required.

The **Properties** menu function on the context menu for the nput/output PROFINET IO modules opens the **Properties** dialog box. The dialog box contains the following tabs:

Tab Parameter

Element	Description
Name	Name of the input/output PROFINET IO modules
Slot	0 to 32767 Default value: 1

Table 36: Parameter Tab of the I/O PROFINET IO Modules

The tabs **Model** and **Features** are self-explanatory providing an additional description and specifying the equipment of the input/output PROFINET IO modules.

5.6.4 Input Submodule

The submodule parameters are used to define the communication relation of the module and its behavior after connection is interrupted.

Properties

The **Properties** function on the context menu for the input submodule opens the **Properties** dialog box. The dialog box contains the following tabs:

Tab Parameter

Element	Description
Name	Name of the input submodule
Sub-Slot	Not changeable Default value: 1
IO Data CR, Inputs	Selection of the communication relation (CR) to which the submodule inputs should be transferred. - None - Default Input CR
Input data accepted by Controller	Selection of the communication relation (CR) to which the submodule IO consumer status (CS) should be transferred. - None - Default Output CR
Shared Input	Activated: Several PROFINET IO controllers can access the inputs. Deactivated: Only one PROFINET IO controller can access the inputs.
Input Values When IO CR is Disconnected	Behavior of the input variables for this PROFINET IO submodule after the connection is interrupted. Retain Last Value: The input variables are freezed to the current value and used until a new connection is established. Adopt Initial Values: The initial data are used for the input variables.

Table 37: Properties Dialog Box for the Input Submodule

The tabs **Model** and **Features** are self-explanatory providing an additional description and specifying the equipment of the input submodule.

Edit

The **Edit** function on the context menu for the input submodule opens the **Edit** dialog box. The dialog box contains the following tabs:

The **System Variables** tab contains the following system variables that are required to evaluate the state of the PROFINET IO submodule from within the user program.

Element	Description
Valid input data	True Valid input data GOOD False Invalid input data BAD_BY_DEVICE, BAD_BY_CONTROLLER
Input data accepted by Controller	True Valid input data GOOD False Invalid input data BAD_BY_SUBSLOT, BAD_BY_SLOT, BAD_BY_DEVICE, BAD_BY_CONTROLLER.

Table 38: Edit Dialog Box for the Input Submodule

The **Process Variables** tab is used to enter the input variables.

5.6.5 Submodule Output

The submodule parameters are used to define the communication relation of the module and its behavior after connection is interrupted.

Properties

The **Properties** function on the context menu for the output submodule opens the **Properties** dialog box. The dialog box contains the following tabs:

Tab Parameter

Element	Description
Name	Name of the output submodule
Sub-Slot	Not changeable Default value: 1
IO Data CR, Outputs	Selection of the communication relation (CR) to which the submodule outputs should be transferred. - None - Default Input CR
Output Data Accepted by Device	Selection of the communication relation (CR) to which the submodule IO consumer status (CS) should be transferred. - None - Default Output CR

Table 39: Properties Dialog Box for the Input Submodule

The tabs **Model** and **Features** are self-explanatory providing an additional description and specifying the equipment of the input submodule.

Edit

The **Edit** function on the context menu for the output submodule opens the **Edit** dialog box. The dialog box contains the following tabs:

The **System Variables** tab contains the following system variables that are required to evaluate the state of the PROFINET IO submodule from within the user program.

Element	Description	
Valid output data	True	Valid output data GOOD
	False	Invalid output data BAD_BY_DEVICE, BAD_BY_CONTROLLER
Output Data Accepted by Device	True	Valid output data GOOD
	False	Invalid output data BAD_BY_SUBSLOT, BAD_BY_SLOT, BAD_BY_DEVICE, BAD_BY_CONTROLLER.

Table 40: Edit Dialog Box for the Output Submodule

The **Process Variables** tab is used to enter the output variables.

5.6.6 Input and Output Submodule

The submodule parameters are used to define the communication relation of the module and its behavior after connection is interrupted.

Properties

The **Properties** function on the context menu for the input/output submodule opens the **Properties** dialog box. The dialog box contains the following tabs:

Tab Parameter

Element	Description
Name	Name of the input/output submodule
Sub-Slot	Not changeable Default value: 1
IO Data CR, Inputs	Selection of the communication relation (CR) to which the submodule inputs should be transferred. - None - Default Input CR
IO Data CR, Outputs	Selection of the communication relation (CR) to which the submodule outputs should be transferred. - None - Default Output CR
Input data accepted by Controller	Selection of the communication relation (CR) to which the submodule IO consumer status (CS) should be transferred. - None - Default Output CR
Output Data Accepted by Device	Selection of the communication relation (CR) to which the submodule IO consumer status (CS) should be transferred. - None - Default Input CR
Input Values When IO CR is Disconnected	- Retain Last Value - Adopt Initial Values

Table 41: Properties Dialog Box for the Input/Output Submodule

The tabs **Model** and **Features** are self-explanatory providing an additional description and specifying the equipment of the input/output submodule.

Edit

The **Edit** function on the context menu for the input/output submodule opens the **Edit** dialog box. The dialog box contains the following tabs:

The **System Variables** tab contains the following system variables that are required to evaluate the state of the PROFINET IO submodule from within the user program.

Element	Description	
Valid output data	True	Valid output data GOOD
	False	Invalid output data BAD_BY_DEVICE, BAD_BY_CONTROLLER
Output Data Accepted by Device	True	Valid output data GOOD
	False	Invalid output data BAD_BY_SUBSLOT, BAD_BY_SLOT, BAD_BY_DEVICE, BAD_BY_CONTROLLER.
Valid input data	True	Valid input data GOOD
	False	Invalid input data BAD_BY_DEVICE, BAD_BY_CONTROLLER
Input data accepted by Controller	True	Valid input data GOOD
	False	Invalid input data BAD_BY_SUBSLOT, BAD_BY_SLOT, BAD_BY_DEVICE, BAD_BY_CONTROLLER.

Table 42: Edit Dialog Box for the Input/Output Submodule

The **Process Variables** tab is used to enter the input and output variables in their corresponding area.

5.6.7 Application Relation

An application relation (AR) is a logic construct for enabling data exchange between controller and device. Data are transferred within the application relation via one or up to five communication relations (CR).

The **Properties** function on the context menu for application relation opens the **Properties** dialog box.

Element	Description
Name	Not changeable
UDP RT Port Controller	UDP port of the controller Not changeable
AR UUID	Code for unambiguously identifying the application relation (AR). Not changeable
Connection Establishment Timeout Factor	From the perspective of the PROFINET IO device, this parameter is used during the creation of a connection to calculate the maximum time allowed between sending the response on the connect request and receiving a new request from the a PROFINET IO controller. Range of values: 1 - 1000 (x 100 ms) Default value: 600
Supervisor may take over the AR	Definition wether a PROFINET IO supervisor may adopt the application relation (AR). Older devices from different manufacturer require this setting. 0 Not Allowed 1 Allowed Default value: 0

Table 43: Properties Dialog Box for the Application Relation

5.6.8 Alarm CR

Multiple communication relations (CR) can be established within an application relation.

The alarm CR is used by a PROFINET IO device to transmit alarms to the PROFINET IO controller.

The **Properties** function on the context menu for application relation opens the **Properties** dialog box. The dialog box contains the following parameters:

Element	Description
Name	Not changeable
VLAN ID, High Priority	Each virtual LAN (VLAN) is assigned a unique number to ensure separation. A device in the VLAN with ID=1 can communicate with any other device in the same VLAN, but not with a device in another VLAN (e.g., ID=2, 3, ...). Range of values: see also IEC 61158-6 0x000 No VLAN 0x001 Standard VLAN 0x002 See IEEE 802.1 Q Up to 0xFFF Default value: 0
VLAN ID, Low Priority	Description, see VLAN ID, High Priority Default value: 0
Alarm Priority	Use User Priority: The priority assigned by the user is used. Ignore User Priority: The priority assigned by the user is ignored. The generated alarm has low priority.
Alarm Resends	Maximum number of device resends if the controller does not respond. Rang of values 3 to 15 Default value: 10
Alarm Timeout Factor	The RTA timeout factor is used to calculate the maximum device time that may elapse after sending a RTA data (alarm) frame and receiving the RTA ack frame. RTA timeout = RTA timeout factor x 100 ms Rang of values 1 to 65535 Default value: 5

Table 44: Properties Dialog Box for the Alarm CR

5.6.9 Input CR

Multiple communication relations (CR) can be established within an application relation.

The input CR is used by a PROFINET IO device to transmit variables to the PROFINET IO controller.

The **Properties** function on the context menu for the default input CR opens the **Properties** dialog box. The dialog box contains the following parameters:

Element	Description
Name	Any unique name for an input CR The default input CR cannot be changed
Type	1 (not changeable)
Send Clock Factor	The send clock factor defines the send clock for the cyclic IO CR data transfer. **Send clock = send clock factor x 31.25 µs** Range of values: 1 to 128 Default value: 32
Reduction Factor	The redundant factor allows the reduction of the actual cycle time needed for sending the data of an IO CR. The actual data cycle time is calculated as follows: **Sending cycle = reduction factor x send clock** Range of values: 1 to 16384 Default value: 32
Watchdog Factor	From the perspective of an IO CR consumer, the watchdog factor is used to calculate the maximum time allowed between the reception of two frames: **Watchdog time = watchdog factor x send clock factor x reduction ratio x 31.25 µs** Range of values: 1 to 7680 Default value: 3
VLAN ID	Each virtual LAN (VLAN) is assigned a unique number to ensure separation. A device in the VLAN with ID=1 can communicate with any other device in the same VLAN, but not with a device in another VLAN (e.g., ID=2, 3, ...). Range of values: see also IEC 61158-6 0x000 No VLAN 0x001 Standard VLAN 0x002 See IEEE 802.1 Q Up to 0xFFF Default value: 0

Table 45: Properties Dialog Box for the Default Input CR

The **Edit** function on the context menu for the default input CR opens the **System Variables** dialog box, and contains the following system variables:

Element	Description
Data Status Input CR	

Table 46: Edit Dialog Box for the Default Input CR

5.6.10 Output CR

Multiple communication relations (CR) can be established within an application relation.

The output CR is used by the PROFINET IO device to transmit variables to the PROFINET IO controller.

The **Properties** function on the context menu for the output CR opens the **Properties** dialog box. The dialog box contains the following parameters:

Element	Description
Name	Any unique name for an output CR The default output CR cannot be changed
Type	2 (not changeable)
Send Clock Factor	The send clock factor defines the send clock for the cyclic IO CR data transfer. **Send clock = send clock factor x 31.25 µs** Range of values: 1 to 128 Default value: 32
Reduction Factor	For setting the transmission frequency. The redundant factor allows the reduction of the actual cycle time needed for sending the data of an IO CR. The actual data cycle time is calculated as follows: **Sending cycle = reduction factor x send clock** Range of values: 1 to 16384 Default value: 32
Watchdog Factor	From the perspective of an IO CR consumer, the watchdog factor is used to calculate the maximum time allowed between the reception of two frames: **Watchdog time = watchdog factor x send clock factor x reduction ratio x 31.25 µs** Range of values: 1 to 7680 Default value: 3
VLAN ID	Each virtual LAN (VLAN) is assigned a unique number to ensure separation. A device in the VLAN with ID=1 can communicate with any other device in the same VLAN, but not with a device in another VLAN (e.g., ID=2, 3, ...). Range of values: see also IEC 61158-6 0x000 No VLAN 0x001 Standard VLAN 0x002 Up to 0xFFF See IEEE 802.1 Q Default value: 0

Table 47: Properties Dialog Box for the Default Output CR

5.7 HIMA PROFINET IO Device

This chapter describes the characteristics of the HIMA PROFINET IO device and the menu functions and dialog boxes required to configure the HIMA PROFINET IO controller in SILworX.

5.8 System Requirements

Equipment and system requirements

Element	Description
Controller	HIMax with COM module
Processor module	The Ethernet interfaces on the processor module may not be used for PROFINET IO .
COM module	Ethernet 10/100BaseT.
Activation	Software activation code required, see Chapter 3.5.

Table 48: Equipment and System Requirements for the PROFINET IO Controller.

PROFINET IO Device Properties

Element	Description
Safety-related	No
Transfer rate	100 Mbit/s full duplex
Transmission path	Ethernet interfaces on the COM module Ethernet interfaces in use can simultaneously be used for additional protocols.
Conformity class	The PROFINET IO device meets the requirements for Conformance Class A.
Max. number of PROFINET IO devices	One PROFINET IO device can be configured for each COM module.
Max. number of application relations (ARs) to the PROFINET IO controller	A PROFINET IO device can establish a maximum of 5 application relations (ARs) to the PROFINET IO controllers or supervisors.
Max. number of communication relations (CRs for each AR)	Max. 5 communication relations (CRs) for each AR und direction
Max. process data length of all configured PROFINET IO modules	Output: max. 1440 bytes Input: max. 1440 bytes
Data Priorization	Possible at device level using the *Reduction Rate* setting.
Interconnecting PROFINET and PROFIBUS	To do this, a PROFINET IO device with proxy functionality is required.

Table 49: PROFINET IO Controller Properties

5.9 PROFINET IO Example

In this example, a HIMA PROFINET IO controller exchanges variables with a HIMA PROFINET IO .device. This example illustrates how to create and configure the HIMA PROFINET IO controller and the HIMA PROFINET IO device.

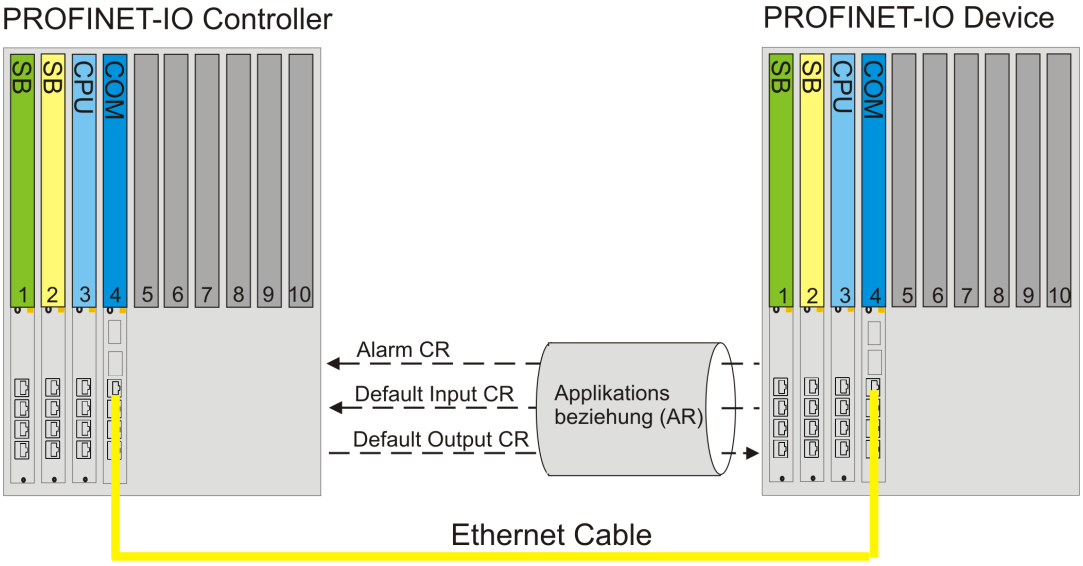


Figure 26: Communication Using PROFINET IO

The communication modules on both HIMax controllers are connected with an Ethernet cable via the Ethernet interface.

An application relation (AR) is a logic construct for enabling data exchange between controller and device. In this example, data are transferred within the application relation via the standard communication relations (alarm CR, default input CR and default output CR). This communication relations are already configured per default in the input and output modules.

For this example, the following global variables must be created in SILworX:

Global Variable	Type
PN_Device_Controller1	UINT
PN_Device_Controller2	DWORD
PN_Device_Controller3	DWORD
PN_Device_Controller4	BYTE
PN_Controller_Device1	UINT
PN_Controller_Device2	BYTE

5.9.1 Configuring the PROFINET IO Device in SILworX

To create a new HIMA PROFINET IO device

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. Select **New**, **PROFINET IO Device** on the context men for protocols to add a new PROFINET IO device.
3. Select **Properties** on the context menu for PROFINET IO controller.
4. Click **COM Module**.

To create the required PROFINET IO Modules

1. In the structure tree, open **Configuration, Resource, Protocols, PROFINET IO Device.**
2. Select **New** on the context menu for the PROFINET IO device.
3. For this example, select the following modules to receive **11 bytes** from the PROFINET IO device and to send **3 bytes**.

PROFINET IO module	Slot
Out 2 Byte_1	1
Out 8 Bytes_2	2
Out 1 Byte_3	3
In 2 Byte_4	4
In 1 Bytes_5	5

To number the PROFINET IO device modules

1. Right-click the first **PROFINET IO device module**, and then click **Properties**.
2. Enter **1** into the **Slot** field.
3. Repeat these steps for every further **PROFINET IO device modules** and number the modules consecutively.

i Number the HIMax PROFINET IO device modules without gaps and in ascending order, starting with **1**.

Configuring the PROFINET IO Device Output Modules

i The sum of the variables (in bytes), must identical with the size of the module (in bytes).

To configure the output module [01] Out 2 Bytes_1

1. In the PROFINET IO device, select the output module **[01] Out 2 Bytes_1**.
2. Right-click **[01] Out 2 Bytes_1** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Output Signals** area.

Name	Type	Offset	Global Variable
PN_Device_Controller1	UINT	0	PN_Device_Controller1

Table 50: Variables in the Output Module [01] Out 2 Bytes_1

5. Right-click anywhere in the **Output Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the output module [02] Out 8 Bytes_2

1. In the PROFINET IO device, select the output module **[02] Out 8 Bytes_2.**

2. Right-click **[02] Out 8 Bytes_2** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Output Signals** area.

Name	Type	Offset	Global Variable
PN_Device_Controller2	DWORD	0	PN_Device_Controller2
PN_Device_Controller3	DWORD	4	PN_Device_Controller3

Table 51: Variables in the Output Module [02] Out 8 Bytes_2

5. Right-click anywhere in the **Output Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the output module [03] Out 1 Bytes_3

1. In the PROFINET IO device, select the output module **[03] Out 1 Bytes_3.**
2. Right-click **[03] Out 1 Bytes_3** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Output Signals** area.

Name	Type	Offset	Global Variable
PN_Device_Controller4	Byte	0	PN_Device_Controller4

Table 52: Variables in the Output Module [03] Out 1 Bytes_3

5. Right-click anywhere in the **Output Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

Configuring the PROFINET IO Device Input Modules

To configure the input module [04] In 2 Bytes_4

1. In the PROFINET IO device, select the input module **[04] Out 2 Bytes_4.**
2. Right-click **[04] In 2 Bytes_4** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area..

Name	Type	Offset	Global Variable
PN_Controller_Device1	UINT	0	PN_Controller_Device1

Table 53: Variables in the Input Module [04] In 2 Bytes_4

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the input module [05] In 1 Bytes_5

1. In the PROFINET IO device, select the input module **[05] In 1 Bytes_5.**
2. Right-click **[05] In 1 Bytes_5** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area..

Name	Type	Offset	Global Variable
PN_Controller_Device2	BYTE	0	PN_Controller_Device2

Table 54: Variables in the Input Module [05] In 1 Byte_5

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To verify the PROFINET IO device configuration

1. In the structure tree, open **Configuration, Resource, Protocols, PROFINET IO Device.**
2. Click the **Verification** button on Action Bar, and then click **OK** to confirm the action.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

i Use the user program of the PROFINET IO device resource to recompile the configuration of the PROFINET IO device and transfer it to the controllers. Only after this step, the new configuration can be used for communication with the PROFINET IO.

5.9.2 Creating a HIMA PROFINET IO Controller in SILworX

To create a new HIMA PROFINET IO controller

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. Select **New**, **PROFINET IO Controller** on the context men for protocols to add a new PROFINET IO controller.
3. Select **Properties** on the context menu for PROFINET IO controller.
4. Click **COM Module**.

Creating a HIMA PROFINET IO Device within the Controller

To create a HIMax PROFINET IO Device within the PROFINET IO controller

1. Select **New**, **PROFINET IO Device** on the context menu for the PROFINET IO controller.

To add the GSDML library file from an external data source (e.g., CD, USB stick, Internet):

1. On the structure tree, select **Configuration**, **Resource**, **Protocols**, **PROFINET IO Controller**, **GSDML Library**.
2. Select **Add GSDML File** on the context menu for the GSDML library and read the GSDML file specific to the PROFINET IO device.

i The GSDML library file usually contains several devices from one manufacturers.
The HIMA GSDML library file is located on the HIMA Web site at: GSDML-V2.1-Hima-embex-20090805.xml.

To load the GSDML file for the PROFINET IO device

1. On the structure tree, select **Configuration**, **Resource**, **Protocols**, **PROFINET IO Controller**, **PROFINET IO Device**.
2. Select **Properties** on the context menu and open the Parameter tab.
   - Enter the IP address of the PROFINET IO device.
   - On the drop-down menu for **GSDML File,** select the GSDML library file specific to PROFINET IO device and close **Properties**.

To select the data access point (DAP) for the PROFINET IO device

1. On the structure tree, select **Configuration**, **Resource**, **Protocols**, **PROFINET IO Controller**, **PROFINET IO Device, DAP Module**.
2. Select **Select Device Access Point (DAP)** on the context menu and choose a suitable data record for the PROFINET IO device.

Creating the HIMax PROFINET IO Controller Modules

The number of bytes that must actually be transferred, must also be configured in the PROFINET IO controller. To do this, add *Modules* until the physical configuration of the device is achieved.

To create the required PROFINET IO Modules

1. On the structure tree, open **Configuration**, **Resource**, **Protocols**, **PROFINET IO Controller**, **PROFINET IO Device**.
2. Select **Insert Modules** on the context menu.
3. For this example, select the following modules to receive **11 bytes** from the PROFINET IO device and to send **3 bytes**.

PROFINET IO module	Slot
DAP Module (Device Access Point)	0
Input 2 Bytes: Module_1	1
Input 8 Bytes: Module_2	2
Input 1 Byte: Module_3	3
Output 2 Bytes: Module_4	4
Output 1 Byte: Module_5	5

To number the PROFINET IO modules

1. Right-click the first **PROFINET IO module**, and then click **Properties**.
2. Enter **0** into the **Slot** field.
3. Repeat these steps for every further **PROFINET IO modules** and number the modules consecutively.

i **Number** the HIMax PROFINET IO modules without gaps and in ascending order, starting with **0**.

Configuring the PROFINET IO Controller Input Modules

i The sum of the variables (in bytes), must identical with the size of the module (in bytes).

For this example, the predefined standard communication relations Default Input CR and Default Output CR are adopted in the submodules of the input and output modules.

To configure the input module [001] Input 2 Bytes: Module_1

1. In the PROFINET IO device, select the input module **[001] Input 2 Bytes: Module_1, [00001] Submodule Inputs_1.**
2. Right-click **[00001] Submodule Inputs_1** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area..

Name	Type	Offset	Global Variable
PN_Device_Controller1	UINT	0	PN_Device_Controller1

Table 55: Variables in the Input Module [001] Input 2 Bytes: Module_1

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the input module [002] Input 8 Byte: Module_2

1. In the PROFINET IO device, select the input module **[002] Input 8 Bytes: Module_2, [00001] Submodule Inputs_1.**
2. Right-click **[00001] Submodule Inputs_1** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area..

Name	Type	Offset	Global Variable
PN_Device_Controller2	DWORD	0	PN_Device_Controller2
PN_Device_Controller3	DWORD	4	PN_Device_Controller3

Table 56: Variables in the Input Module [002] Input 8 Byte: Module_2

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the input module [003] Input 1 Byte: Module_3

1. In the PROFINET IO device, select the input module **[003] Input 1 Byte: Module_1, [00001] Submodule Inputs_1**.
2. Right-click **[00001] Submodule Inputs_1** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area.

Name	Type	Offset	Global Variable
PN_Device_Controller4	Byte	0	PN_Device_Controller4

Table 57: Variables in the Input Module [003] Input 1 Byte: Module_3

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

Configuring the PROFINET IO Controller Output Modules

To configure the output module [004] Output 2 Bytes: Module_4

1. In the PROFINET IO device, select the output module **[004] Output 2 Bytes: Module_4, [00001] Submodule Inputs_1**.
2. Right-click **[00001] Submodule Inputs_1** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Output Signals** area.

Name	Type	Offset	Global Variable
PN_Controller_Device1	UINT	0	PN_Controller_Device1

Table 58: Variables in the Output Module [004] Out 2 Bytes: Module_4

5. Right-click anywhere in the **Output Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the output module [005] Output 1 Bytes: Module_5

1. In the PROFINET IO device, select the output module **[005] Output 1 Byte: Module_5, [00001] Submodule Inputs_1**.
2. Right-click **[00001] Submodule Inputs_1** and select **Edit** on the context menu.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Output Signals** area.

Name	Type	Offset	Global Variable
PN_Controller_Device2	BYTE	0	PN_Controller_Device2

Table 59: Variables in the Output Module [005] Out 1 Bytes: Module_5

5. Right-click anywhere in the **Output Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

5.9.3 Menu Function Properties

The **Properties** menu function on the context menu for the PROFINET IO device opens the **Properties** dialog box.

Element	Description
Type	PROFINET IO device
Name	Any unique name for a PROFINET IO device
Refresh Rate [ms]	Refresh rate in milliseconds at which the COM and CPU exchange protocol data. If the *Refresh Rate* is zero or lower than the cycle time for the controller, data is exchanged as fast as possible. Range of values: 4...(2^{31}-1). Default value: 0
Within one cycle	Activated: Transfer of all protocol data from the CPU to the COM within a CPU cycle. Deactivated: Transfer of all protocol data from the CPU to the COM, distributed over multiple CPU cycles, each with 1100 byte per data direction. This can also allow lowering the cycle time of the controller. Default value: Activated
Module	Selection of the COM module within which the protocol is processed.
Use Max CPU Load	Activated: Use CPU load limit from the field *Max. CPU Load [%]* Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%

RPC Port Server	Remote Procedure Call Port Range of values: 1024...65535 Default value: 49152 RPC port server and RPC port client must not be identical!
RPC Port Client	Remote Procedure Call Port Range of values: 1024...65535 Default value: 49153 RPC port server and RPC port client must not be identical!
RT Port Controller	RT Port Range of values: 1024...65535 Default value: 34962

Table 60: PROFINET IO Device General Properties

5.9.4 PROFINET IO Modules

The following PROFINET IO modules are available in the HIMA PROFINET IO device.

PROFINET IO module	Max. size for the input variables	Max. size for the output variables
In 1 byte	1 byte	
In 2 bytes	2 bytes	
In 4 bytes	4 bytes	
In 8 bytes	8 bytes	
In 16 bytes	16 bytes	
In 32 bytes	32 bytes	
In 64 bytes	64 bytes	
In 128 bytes	128 bytes	
In 256 bytes	256 bytes	
In 512 bytes	512 bytes	
In 1024 bytes	1024 bytes	
In-Out 1 byte	1 byte	1 byte
In-Out 2 bytes	2 bytes	2 bytes
In-Out 4 bytes	4 bytes	4 bytes
In-Out 8 bytes	8 bytes	8 bytes
In-Out 16 bytes	16 bytes	16 bytes
In-Out 32 bytes	32 bytes	32 bytes
In-Out 64 bytes	64 bytes	64 bytes
In-Out 128 bytes	128 bytes	128 bytes
In-Out 256 bytes	256 bytes	256 bytes
In-Out 512 bytes	512 bytes	512 bytes
In-Out 1024 bytes	1024 bytes	1024 bytes
Out 1 byte		1 byte
Out 2 bytes		2 bytes
Out 4 bytes		4 bytes
Out 8 bytes		8 bytes
Out 16 bytes		16 bytes
Out 32 bytes		32 bytes
Out 64 bytes		64 bytes
Out 128 bytes		128 bytes
Out 256 bytes		256 bytes
Out 512 bytes		512 bytes
Out 1024 bytes		1024 bytes

Table 61: PROFINET IO Device General Properties

To create a PROFINET IO module

1. In the structure tree, open **Configuration, Resource, Protocols, PROFINET IO Device.**
2. Select **New** on the context menu for the PROFINET IO device.
3. Right-click PROFINET IO module and select **Edit**.
   - Enter the input and/or output variables in the **Process Variables** tab.
   - The **System Variables** can be used to assign global variables to both system variables and use them in the user program

Element	Type
Valid output data	BOOL
Accept the output data from the controller	BOOL

   - The **Properties** tab specifies the following parameters.

Element	Description
Name	Name of the PROFINET IO device module
Slot	0 to 32767
Module ID	Unique number
IO type	1 In 2 Out 3 In-Out
Process data after termination	Process data value after the connection is interrupted - Retain last valid process data - Adopt initial data
Length of IO Input Data	0 ...1024
Length of IO Output Data	0 ...1024

Table 62: PROFINET IO Device General Properties

6 PROFIBUS DP

PROFIBUS DP is an international, open fieldbus standard that is used when a fast reaction time is required for small amounts of data.

The HIMA PROFIBUS DP master and the HIMA PROFIBUS DP slave meet the criteria specified in the European norm EN 50170 [7] and the globally binding IEC standard 61158 for PROFIBUS DP.

The HIMA PROFIBUS DP master can exchange data with the PROFIBUS DP slaves cyclically and acyclically.

Different function blocks are available in SILworX to acyclically exchange data. These function blocks are used to tailor the HIMA PROFIBUS DP master and the PROFIBUS DP slaves to best meet the project requirements.

A redundant PROFIBUS DP connection can only be implemented by configuring a second PROFIBUS DP master/slave and adjusting it in the user program.

- PROFIBUS DP master (see Chapter 6.1)
- PROFIBUS DP slave (see Chapter 6.13)

6.1 HIMA PROFIBUS DP Master

This chapter describes the characteristics of the HIMA PROFIBUS DP master and the menu functions and dialog boxes required to configure the HIMA PROFIBUS DP master in SILworX.

Equipment and System Requirements:

Element	Description
HIMA controller	HIMax with COM module
COM module	The serial fieldbus interface (FB1 or FB2) used on the COM module must be equipped with an optional HIMA PROFIBUS DP master submodule, see Chapter 3.7.
Activation	Activation through the plug-in module, see Chapter 3.5.

Table 63: Equipment and System Requirements

PROFIBUS DP Master Properties:

Element	Description
Type of HIMA PROFIBUS DP master	DP-V1 Class 1 Master with additional DP-V2 functions
Transfer rate	9.6 kbit/s ... 12 Mbit/s
Bus address	0 ...125
Max. number of PROFIBUS DP master	Two PROFIBUS DP masters can be configured for each COM module.
Max. number of PROFIBUS DP slaves	Up to 122 slaves can be configured for each resource (in all master protocol instances). However, a maximum of 31 slaves can be connected to a bus segment without repeaters.
Maximum process data length To a slave	DP output=: max. 244 bytes DP input=: max. 244 bytes

Table 64: PROFIBUS DP Master Properties

According to the standard, a total of three repeaters may be used such that a maximum of 122 stations are possible per serial interface on a master.

6.1.1 Creating a HIMA PROFIBUS DP Master

To create a new HIMA PROFIBUS DP Master

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, click **New, PROFIBUS DP Master** to add a new PROFIBUS DP master.
3. On the context menu for the PROFIBUS DP master, click **Properties, General**.
4. Select **Module** and **Interfaces**.

6.2 PROFIBUS DP: Example

In this example, a HIMA PROFIBUS DP master exchanges variables with a HIMA PROFIBUS DP slave.

The example shows how to create and configure the HIMA PROFIBUS DP master and the HIMA PROFIBUS DP slave.

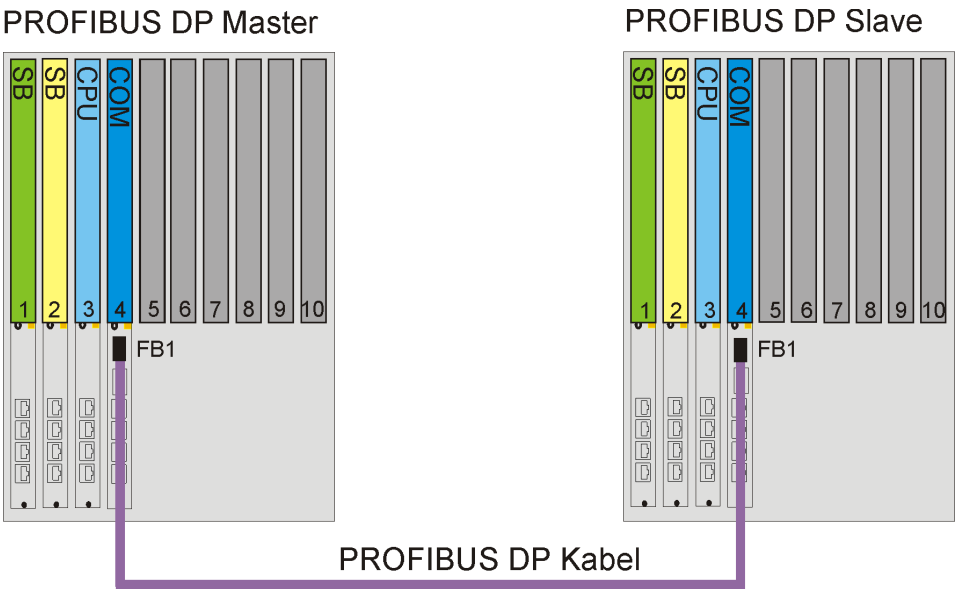


Figure 27: Communication Using PROFIBUS DP

Fieldbus Interface 1 on the COM modules of both HIMax controllers must be equipped with the corresponding PROFIBUS DP submodule, see Chapter 3.7.

For this example, the following global variables must be created in SILworX:

Global Variable	Type
PB_Slave_Master1	UINT
PB_Slave_Master2	DWORD
PB_Slave_Master3	DWORD
PB_Slave_Master4	BYTE
PB_Master_Slave1	DWORD
PB_Master_Slave2	BYTE

6.2.1 Configuring the PROFIBUS DP Slave

Configuration of the PROFIBUS DP slave.

To create a new HIMA PROFIBUS DP Slave

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, click **New, PROFIBUS DP Slave** to add a new PROFIBUS DP slave.
3. On the context menu for the PROFIBUS DP slave, click **Edit**.
4. In the **Properties** tab, select **COM Module** and **Interfaces** (e.g., FB1).

To assign variables in the HIMA PROFIBUS DP slave

1. On the context menu for the PROFIBUS DP slave, click **Edit**.
2. In the **Edit** dialog box, select the **Process Variables** tab.

i The start address of the input and output variables in the HIMA PROFIBUS DP slave always begin with 0. If the PROFIBUS DP master (from another manufacturer) expects a higher start address, dummy variables must be added to the reference variables.

Outputs in the HIMA PROFIBUS DP slave

Name	Type	Offset	Global Variable
PB_Slave_Master1	UINT	0	PB_Slave_Master1
PB_Slave_Master2	DWORD	2	PB_Slave_Master2
PB_Slave_Master3	DWORD	6	PB_Slave_Master3
PB_Slave_Master4	BYTE	10	PB_Slave_Master4

Table 65: Outputs in the HIMA PROFIBUS DP Slave

1. Drag the global variable to be sent from the Object Panel onto the **Output Variables** area.

i In this example, the output variables of the HIMA PROFIBUS DP slave are composed of **four variables** with a total of **11 bytes**. The start address of the output variable with the lowest offset is **0**.

2. Right-click anywhere in the **Output Variables** area to open the context menu.
3. Click **New Offsets** to re-generate the variable offsets.

Inputs in the HIMA PROFIBUS DP Slave

Name	Type	Offset	Global Variable
PB_Master_Slave1	DWORD	14	PB_Master_Slave1
PB_Master_Slave2	BYTE	18	PB_Master_Slave2

Table 66: Inputs in the HIMA PROFIBUS DP Slave

1. Drag the global variables to be received from the Object Panel onto the **Input Variables** area.

i In this example, the input variables of the HIMA PROFIBUS DP slave are composed of **two variables** with a total of **3 bytes**. The start address of the input variable with the lowest offset is **0**.

2. Right-click anywhere in the the **Input Variables** area to open the context menu.
3. Click **New Offsets** to re-generate the variable offsets.

To verify the configuration of the PROFIBUS DP slave

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Slave.**

2. Click the **Verification** button on Action Bar, and then click **OK** to confirm the action.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

i Use the user program of the PROFIBUS DP slave to recompile the configuration of the PROFIBUS DP slave resource and transfer it to the controllers. Only after this step, the new configuration can be used for communication with the PROFIBUS DP.

6.2.2 Configuring the PROFIBUS DP Master

To create a new HIMA PROFIBUS DP Master

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. Select **New, PROFIBUS DP Master o**n the context menu for protocols to add a new PROFIBUS DP master.
3. Select **Properties, General** on the context menu for the PROFIBUS DP master.
4. In the **General** tab, select **COM Module** and **Interfaces** (e.g., FB1).

i Perform these steps to configure the HIMax PROFIBUS DP slave from within the HIMax PROFIBUS DP master.

To create a new HIMax PROFIBUS DP Slave in the PROFIBUS DP Master

1. On the context menu for the PROFIBUS DP master, click **New, PROFIBUS DP Slave**.

To read the GSD file for the new PROFIBUS DP slave

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, PROFIBUS Slave**.
2. On the context menu for the PROFIBUS DP master, click **Read GSD File** and select the GSD file for the PROFIBUS slave (e.g., hax100ea.gsd).

i The GSD files for HIMax controllers are available on HIMA website at www.hima.com.

Creating the HIMax PROFIBUS DP Modules

The number of bytes that must actually be transferred, must also be configured in the PROFIBUS DP master. To do this, add *Modules* until the physical configuration of the slave is achieved.

i The number of modules used to achieve the necessary number of bytes is not important as long as the maximum of 32 modules is not exceeded.

To avoid unnecessarily complicating the PROFIBUS DP master configuration, HIMA recommends keeping the number of selected modules to a minimum.

To create the required PROFIBUS DP Modules

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, PROFIBUS Slave**.
2. On the menu bar, click **PROFIBUS DP Master, Add modules**.
3. For this example, select the following modules to receive **11 bytes** from the PROFIBUS DP slave and to send 3 bytes.

To number the PROFIBUS DP modules

1. Right-click the first **PROFIBUS DP Module** , and then click **Properties**.
2. Enter **0** into the **Slot** field.
3. Repeat these steps for every further **PROFIBUS DP Module** and number the modules consecutively.

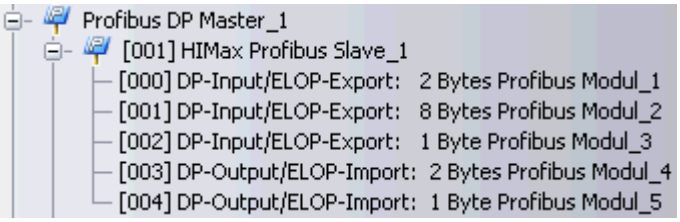


Figure 28: HIMax PROFIBUS DP Slave with Modules

i **Number** the HIMax PROFIBUS DP modules without gaps and in ascending order, starting with **0**.

The order in which the PROFIBUS DP modules are arranged is not important for operation. However, HIMA recommends organizing the DP input and output modules in an orderly manner to ensure an overview can be maintained.

Configuring the Input and Output Modules

i The sum of the variables (in bytes), must identical with the size of the module (in bytes).

To configure the input module [000] DP Input/ELOP Export: 2 bytes

1. In the PROFIBUS DP slave, select the input module **[000] DP Input/ELOP Export: 2 Bytes**
2. Right-click the input module, then click **Edit**.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area of the input module **[000] DP-Input/ELOP-Export: 2 Bytes**.

Name	Type	Offset	Global Variable
PB_Slave_Master1	UINT	0	PB_Slave_Master1

Table 67: Variables of the Input Module [000] DP Input/ELOP Export: 2 Bytes

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the input module [001] DP Input/ELOP Export: 8 bytes

1. In the PROFIBUS DP slave, select the input module **[001] DP Input/ELOP Export: 8 Bytes**
2. Right-click the input module, then click **Edit**.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area of the input module **[001] DP-Input/ELOP-Export: 8 Bytes**.

Name	Type	Offset	Global Variable
PB_Slave_Master2	DWORD	0	PB_Slave_Master2
PB_Slave_Master3	DWORD	4	PB_Slave_Master3

Table 68: Variables of the Input Module [001] DP Input/ELOP Export: 8 Bytes

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the input module [002] DP Input/ELOP Export: 1 byte

1. In the PROFIBUS DP slave, select the input module **[002] DP Input/ELOP Export: 1 Byte**.
2. Right-click the input module, then click **Edit**.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Input Signals** area of the input module [**002] DP-Input/ELOP-Export: 1 Byte**.

Name	Type	Offset	Global Variable
PB_Slave_Master4	BYTE	0	PB_Slave_Master4

Table 69: Variables of the Input Module [002] DP Input/ELOP Export: 1 Byte

5. Right-click anywhere in the **Input Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the output module [003] DP Output/ELOP Import: 2 Bytes

1. In the PROFIBUS DP slave, select the output module **[003] DP Output/ELOP Import: 2 Bytes**
2. Right-click the output module, then click **Edit**.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Output Signals** area of the output module **[003] DP-Output/ELOP-Import: 2 Bytes** .

Name	Type	Offset	Global Variable
PB_Master_Slave1	UINT	0	PB_Master_Slave1

Table 70: Variables of the Output Module [003] DP Output/ELOP Import: 2 Bytes

5. Right-click anywhere in the **Output Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

To configure the output module [004] DP Output/ELOP Import: 1 Byte

1. In the PROFIBUS DP slave, select the output module **[004] DP Output/ELOP Import: 1 Byte**
2. Right-click the output module, then click **Edit**.
3. In the **Edit** dialog box, select the **Process Variables** tab.
4. Drag the suitable variable from the Object Panel onto the **Output Signal**s area of the output module **[004] DP-Output/ELOP-Import: 1 Byte**.

Name	Type	Offset	Global Variable
PB_Master_Slave2	BYTE	0	PB_Master_Slave2

Table 71: Variables of the Output Module [004] DP Output/ELOP Import: 1 Byte

5. Right-click anywhere in the **Output Signals** area to open the context menu.
6. Click **New Offsets** to re-generate the variable offsets.

Creating the User Data in the PROFIBUS DP Master

To create the user data in the PROFIBUS DP master

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master.**
2. Right-click **PROFIBUS Master** , and then click **Properties**.
3. Select the **Data** tab and click the **Buttons ...** next to the user data.

In the 32 bytes long user data field the group's start address and the *block's number of variables* are defined (see also chapter 6.8).

4. For this example, create the following user data:
   4, to ensure that **four variables** are received by the PROFIBUS DP master.
   2, to ensure that two variables that are sent by the PROFIBUS DP master.
   The start address of the input and output groups begins with **0**.

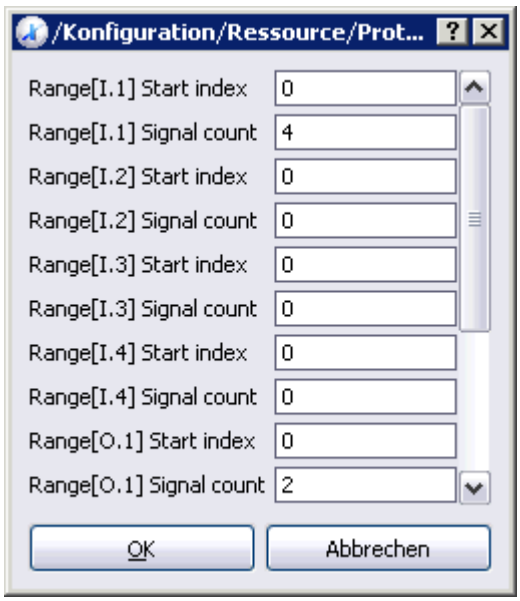


Figure 29: User Data Field

To verify the configuration of the PROFIBUS DP slave

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master**.
2. Click the **Verification** button on Action Bar, and then click **OK** to confirm the action.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

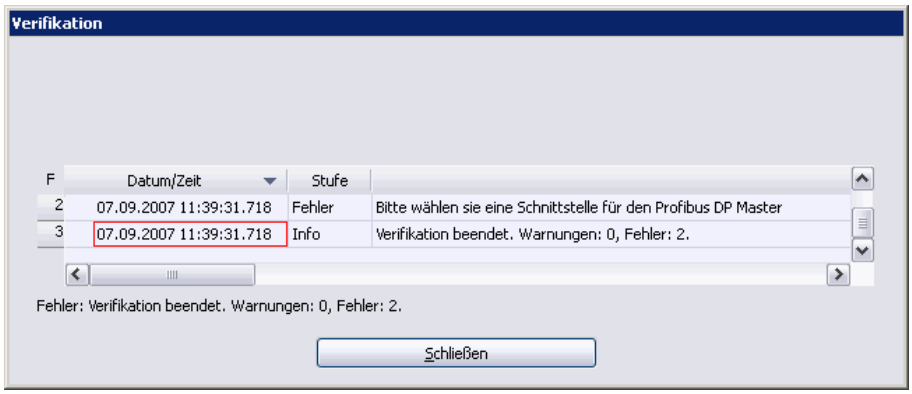


Figure 30: Verification Dialog Box

i Use the user program of the PROFIBUS DP master to recompile the configuration of the PROFIBUS DP master resource and transfer it to the controllers. Only after this step, the new configuration can be used for communication with the PROFIBUS DP

Optimizing the PROFIBUS DP parameters

Using the default values for the PROFIBUS parameters, smooth PROFIBUS communication is generally not a problem. However, the settings should be further optimized to achieve faster data exchange rates and improve fault detection.

To determine the actual target rotation time TTR [ms]

1. Open the Control Panel associated with the HIMax PROFIBUS DP master controller.
2. In the structure tree for the Control Panel, click **PROFIBUS DP Master** and read the actual **Target Rotation Time TTR [ms]**. Note down this value.

To determine the parameters required for the PROFIBUS DP slave

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, PROFIBUS DP Slave**..
2. Right-click **HIMax PROFIBUS Slave** , and then click **Properties**.
3. Select the **Features** tab and read **Min. Slave Interval MSI [ms]** for this PROFIBUS DP slave. Note down this value.
4. Select the **Transfer Rate** tab and read **Max. Tsdr** for the transfer rate used. Note down this value.

To enter the parameters previously determined

1. Right-click **PROFIBUS Master** , and then click **Properties**.
2. Select the **Timings** tab.

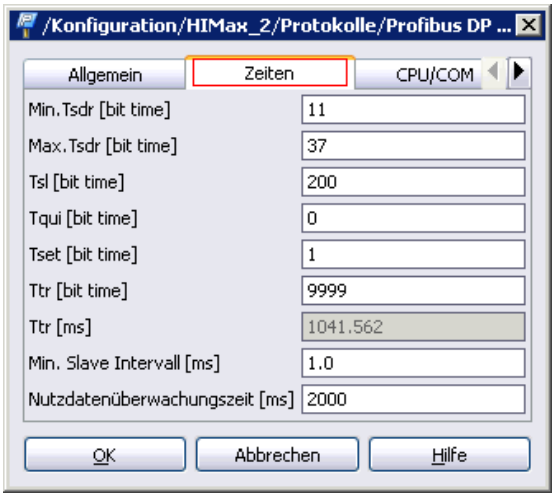


Figure 31: PROFIBUS DP Master Properties

3. Convert the **Max. Tsdr** that was previously noted down in **bit Time**
4. Convert the **Target Rotation Time TTR [ms]** that was previously noted down in **bit Time**, add 1/3 safety margin and enter the resulting value in the **Target Rotation Time TTR [ms]** field.
5. Enter the **Min. Slave Interval MSI [ms]** that was previously noted down.

i If various slaves are configured, the highest values of the parameters MaxTsdr [bit time] and Min. Slave Interval [ms] must be used.

6. The data control time [ms] must be set to ≥ 6*Ttr ms

To enter the watchdog time for the PROFIBUS DP slave

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, PROFIBUS DP Slave**..
2. Right-click **HIMax PROFIBUS Slave** , and then click **Properties**.

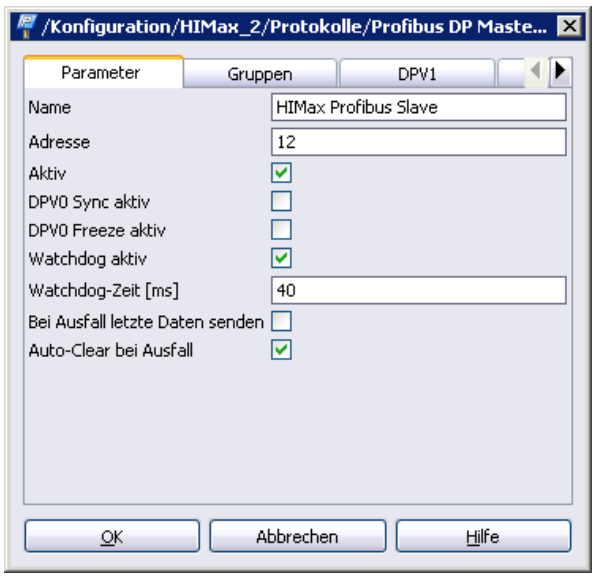


Figure 32: PROFIBUS DP Slave Properties

3. Select the **Parameter** tab and mark the **Watchdog Active** checkbox.
4. Enter the watchdog time [ms] ≥ 6*Ttr [ms] in the **Watchdog Time [ms]** field.

i Use the user program of the PROFIBUS DP master and slave resources to recompile the configurations of the PROFIBUS DP master and slave and transfer them to the controllers. Only after this step, the new configurations can be used for communication with the PROFIBUS DP

6.3 Menu Functions of the PROFIBUS DP Master

6.3.1 Edit

The **Edit** function on the context menu for the PROFIBUS DP master opens the **Edit** dialog box.

The **System Variables** tab contains the following system variables that are required to evaluate the state of the PROFIBUS DP master from within the user program.

<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>Error count</td><td>Number of errors since statistics reset.</td></tr>
<tr><td>Baud rate</td><td>Baud rate (bit/s) used for the bus.</td></tr>
<tr><td>Bus error</td><td>If a bus error occurs, an error code is set in the Bus Error system variable. An error code retains its value until the bus error has been eliminated.
<table>
<tr><th>Code</th><th>Description</th></tr>
<tr><td>0</td><td>OK, no bus error</td></tr>
<tr><td>1</td><td>Address error:
The master address is already available on the bus</td></tr>
<tr><td>2</td><td>Bus malfunction
Malfunction detected on the bus, (e.g., bus not properly terminated, several stations are sending data simultaneously).</td></tr>
<tr><td>3</td><td>Protocol errors
An incorrectly coded packet was received.</td></tr>
<tr><td>4</td><td>Hardware fault
The hardware reported a fault, e.g., too short time periods.</td></tr>
<tr><td>5</td><td>Unknown error
The master changed the status for an unknown reason.</td></tr>
<tr><td>6</td><td>Controller Reset
The controller chip is reset if a serious bus error occurs.</td></tr>
</table>
To evaluate the Bus Error status variable from within the user program, it must be connected to a variable.</td></tr>
<tr><td>Average cycle time</td><td>Measured average bus cycle time in milliseconds.</td></tr>
<tr><td>Last cycle time</td><td>Measured bus cycle time in milliseconds.</td></tr>
<tr><td>Master State</td><td>Indicate the current protocol state.
0: OFFLINE
1: STOP
2: CLEAR
3: OPERATE
Connect the status variable Master Status to a variable to evaluate it in the user program.</td></tr>
<tr><td>Maximum Cycle Time</td><td>Measured maximum bus cycle time in milliseconds.</td></tr>
<tr><td>Min. Slave Interval</td><td>Minimum slave interval measured for one of the slaves assigned to this master.</td></tr>
<tr><td>Minimum Cycle Time</td><td>Measured minimum bus cycle time in milliseconds.</td></tr>
<tr><td>Target Rotation Time</td><td>Target token rotation time</td></tr>
</table>

Table 72: System Variables in the PROFIBUS DP Master

6.3.2 Menu Function 'Properties'

The **Properties** function on the context menu for the PROFIBUS DP master opens the **Properties** dialog box.

The dialog box contains the following tabs:

Tab General

<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>Type</td><td>PROFIBUS DP master</td></tr>
<tr><td>Name</td><td>Any unique name for a PROFIBUS DP Master</td></tr>
<tr><td>Module</td><td>Selection of the COM module within which the protocol is processed.</td></tr>
<tr><td>Use Max CPU Load</td><td>Activated:
Use CPU load limit from the field Max. CPU Load [%]
Deactivated:
Do not use the CPU Load limit for this protocol.</td></tr>
<tr><td>Max. CPU Load [%]</td><td>Maximum CPU load of module that can be used for processing the protocols.

Range of values: 1...100%
Default value: 30%</td></tr>
<tr><td>Address</td><td>Master station address.
Only one master station address may be available on the bus.

Range of values: 0...125
Default value: 0</td></tr>
<tr><td>Interface</td><td>COM interface that should be used for the master.
Range of values: FB1, FB2</td></tr>
<tr><td>Baud Rate</td><td>Baud rate (bit/s) used for the bus.
Possible values:
<table>
<tr><th>Value</th><th>Baud Rate</th><th>FB1</th><th>FB2</th></tr>
<tr><td>9600</td><td>9.6 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>19200</td><td>19.2 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>45450</td><td>45.45 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>93750</td><td>93.75 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>187500</td><td>187.5 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>500000</td><td>500 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>1500000</td><td>1.5 Mbit/s</td><td>X</td><td>X</td></tr>
<tr><td>3000000</td><td>3 Mbit/s</td><td>X</td><td>-</td></tr>
<tr><td>6000000</td><td>6 Mbit/s</td><td>X</td><td>-</td></tr>
<tr><td>12000000</td><td>12 Mbit/s</td><td>X</td><td>-</td></tr>
</table></td></tr>
</table>

Table 73: General Properties for PROFIBUS DP Master

Tab Timings

Element	Description
MinTsdr [bit time]	Min. Station Delay Time: Minimum time period that a PROFIBUS DP slave must wait before it may respond. Range of values: 11 ... 1023 Default value: 11
MaxTsdr [bit time]	Max. Station Delay Time: Maximum time period that a PROFIBUS DP slave may need to respond. Max Tsdr ≥ Tsdr (of the connected slave with the highest Tsdr) The MaxTsdr values of the slaves are read from the GSD files and are displayed in the **Baud rates** tab located in the slave's **Properties** dialog box range of values: 37 ... 65525 Default value: 37
Tsl [bit time]	Slot Time Maximum time span that the master waits for a slave's acknowledgment. Tsl > MaxTsdr + 2*Tset +Tqui + 13 Range of values: 37...16383 Default value: 37
Tqui [bit time]	Quiet Time for Modulator Time that a station may need to switch from sending to receiving. Range of values: 0...493 Default value: 0
Tset [bit time]	Setup Time Time for reacting to an event. Range of value: 1 ... 494 Default value: 1
Ttr [bit time]	Time configured for a token cycle. Maximum time available for a token rotation. A lower estimate of the Ttr can be obtained with a specific calculation, see Chapter 6.4.4. Range of values: 256 ... 16777215 Default value: 999
Ttr [ms]	Actual token rotation time in ms
Min. Slave Interval [ms]	Minimum time between two cyclical requests of a slave. The master observes the Min. Slave Interval and does not fall below it. However, the PROFIBUS DP cycle can be extended if Isochronous Mode is inactive and the portion of acyclic telegrams increases within a cycle. The value for the Min. Slave Interval is read from the GSD file and appears in the *Features* tab located in the **Properties** dialog box. In Isochronous Mode, the value for *Min. Slave Interval* defines the time period for an isochronous cycle.
Element	Description
Min. Slave Interval [ms] (continuation)	Isochronous Mode is activated if the options Isochronous Sync Mode or Isochronous Freeze Mode are activated. See also Refresh Rate between CPU and COM (CPU/COM tab). Range of values: 0...6553,5 Default value: 1,0
Data Control Time [ms]	Time span within which the master must report its current state on the bus Standard value Standard value: Data Control Time = WDT of the slave Range of values: 0...65535 [10 ms] Default value: 2000

Table 74: Timings Tab in the Properties Dialog Box for the PROFIBUS DP Master

Tab CPU/COM

The default values of the parameters provide the fastest possible data exchange of PROFIBUS DP data between the COM module (COM) and the processor module (CPU) within the HIMax controller. These parameters should only be changed if it is necessary to reduce the COM or CPU load for an application, and the process allows this change.

i Only experienced programmers should modify the parameters. Increasing the COM and CPU refresh rate means that the effective refresh rate of the PROFIBUS DP data is also increased. The system time requirements must be verified.

Take also the parameter *Min. Slave Interval [ms]* into account which defines the refresh rate of the PROFIBUS DP data from/to the PROFIBUS DP slave. The refresh rate of the PROFIBUS DP data can be increased according to the CPU/COM refresh rate.

Element	Description
Refresh Rate [ms]	Refresh rate in milliseconds at which the COM and CPU exchange protocol data. If the *Refresh Rate* is zero or lower than the cycle time for the controller, data is exchanged as fast as possible. Range of values: 0...(2^{31}-1). Default value: 0
Within one cycle	Activated: Transfer of all protocol data from the CPU to the COM within a CPU cycle. Deactivated: Transfer of all protocol data from the CPU to the COM, distributed over multiple CPU cycles, each with 1100 byte per data direction. This can also allow lowering the cycle time of the controller. Default value: Activated

Table 75: CPU/COM Tab in the Properties Dialog Box for the PROFIBUS DP Master

Tab Other

Element	Description
Max. number of Resends	Maximum number of resends attempted by a master if a slave does not respond. Range of values: 0...7 Default value: 1
Highest Active Address	Highest Station Address (HSA) Highest station address to be expected for one master. Masters having a station address beyond the HSA are not included into the token ring. Range of values: 0...125 Default value: 125

Isochronous Sync Mode	Isochronous Sync Mode allows both a clock-controlled synchronization of the master and the slaves and a simultaneous activation of the physical outputs of several slaves. If Isochronous Sync Mode is active, the master sends the "Sync" control command as a broadcast telegram to all slaves. As soon as the slaves supporting Isochronous Sync Mode receive the "Sync" control command, they synchronously switch the data from the user program to the physical outputs. The physical outputs' values remain frozen up to the next Sync control command. The cycle time is determined by the "Min. Slave Interval". Condition: Ttr < Min. Slave Interval Default value: Deactivated
Isochronous Freeze Mode	Isochronous Freeze Mode allows the user to simultaneously accept the input data of several slaves. If Isochronous Freeze Mode is active, the master sends the "Freeze" control command as a broadcast telegram to all slaves. As soon as the slaves supporting Isochronous Freeze Mode receive the "Freeze" control command , the physical inputs' variables are frozen to the current value. The master can thus read the values. The input data is only updated when the next "Freeze" control command is sent. The cycle time is determined by the "Min. Slave Interval". Condition: Ttr < Min. Slave Interval Default value: Deactivated
Auto clear on error	If "Auto clear on error" is set in a slave that fails, the master adopts the CLEAR state. Default value: Deactivated
Time Master	The master is also time master and periodically sends the system time via the bus. Default value: Deactivated
Clock Sync Interval [ms]	Clock Synchronization Interval. Time interval within which the time master sends the system time over the bus. Range of values: 0 ... 65535 Default value: 0

Table 76: Other Properties for the PROFIBUS DP Master

6.4 PROFIBUS DP Bus Access Method

The bus access method provides a defined time window to every station within which the station can perform its communication tasks.

6.4.1 Master/Slave Protocol

The bus assignment between a PROFIBUS DP master and a PROFIBUS DP slave is ensured by the master/slave method.

An active PROFIBUS DP master communicates with passive PROFIBUS DP slaves.

The PROFIBUS DP master with the token is authorized to send and may communicate with the PROFIBUS DP slaves assigned to it. The master assigns the bus to a slave for a certain time and the slave must respond within this time period.

6.4.2 Token Protocol

The bus assignment between automation devices (class 1 masters) and/or programming devices (class 2 masters) is ensured via token passing.

All PROFIBUS DP masters connected to a common bus form a token ring. As long as the active PROFIBUS DP master has the token, it assumes the master function on the bus.

In a token ring, the PROFIBUS DP masters are organized in ascending order according to their station addresses. The token is passed on in this order until it is received by the PROFIBUS DP master with the highest station address

This master passes the token on to the master with the lowest station address to close the token ring.

The token rotation time corresponds to one token cycle through all the PROFIBUS DP masters. The target rotation time (Ttr) is the maximum time permitted for a token cycle.

6.4.3 Target Token Rotation Time (Ttr)

Default Values for different transfer rates

While configuring the PROFIBUS DP master, take into account that some parameters set in the **Timings** tab depend on the baud rate set in the **General** tab. For the first (initial) configuration, use the default values specified in the following table. The values are optimized in a later step.

	9.6k	19.2k	45.45k	93.75k	187.5k	500k	1.5M	3M	6M	12M
MinTsdr	11	11	11	11	11	11	11	11	11	11
MaxTsdr	60	60	400	60	60	100	150	250	450	800
Tsl bit time	100	100	640	100	100	200	300	400	600	1000
Tqui bit time	0	0	0	0	0	0	0	3	6	9
Tset bit time	1	1	95	1	1	1	1	4	8	16

Table 77: HIMax Default Values for Token Rotation Time Used with Different Transfer Rates

All time values specified are expressed in Tbit (1Tbit = 1/[bit/s]).

MinTsdr is at least 11 Tbits long as a character has 11 bits (1 start bit, 1 stop bit, 1 parity bit, 8 data bits).

Transmission Time for a Character

Baud Rate	Tbit bit = 1/baud rate	Time
9600 bit/s	1 / 9600 = 104.166 µs	11*104.166 µs = of 114.583 ms
6 Mbit/s	1/ 6*1066 = 166.667 ns	11*166.667 ns = 1.833 µs

Table 78: Transmission Time for a Character Used with different Transfer Rates

6.4.4 Calculating the Target Token Rotation Time (Ttr)

Calculate the minimum target token rotation time Ttr as follows:

Ttr_{min} = n * (198 + T1 + T2) + b * 11 + 242 + T1 + T2 + Tsl

Element	Description
n	Number of active slaves
b	Number of I/O data bytes of the active slaves (input plus output)
T0	35 + 2 * Tset + Tqui
T1	If T0 < MinTsdr: T1 = MinTsdr If T0 > MinTsdr: T1 = T0
T2	If T0 < MaxTsdr: T2 = MaxTsdr If T0 > MaxTsdr: T2 = T0
Tsl	Slot Time: Maximum time period that the master waits for a slave to respond
198	Twice a telegram header with variable length (for request and response)
242	Global_Control, FDL_Status_Req and token passing

Table 79: Elements Required for Calculating the Target Token Rotation Time

i The estimate of the token rotation time *Ttr* is only valid if the following conditions are met: only one master is operating on the bus, no transmissions are repeated and no acyclic data is transmitted.

Never set *Ttr* to a value less than that calculated with the above formula. Otherwise fault-free operation can no longer be ensured. HIMA recommends using a value two or three times greater than the result.

Example of Calculating the Token Rotation Time *Ttr*

The following configuration is available:

5 active slaves

(n = 5)

20 I/O data bytes per slave

(b = 100)

The following time constants for a transmission rate of 6 Mbit/s are taken from Table 79:

- MinTsdr = 11 T_{bit}
- MaxTsdr = 450 T_{bit}
- Tsl bit time = 600 T_{bit}
- Tqui bit time = 6 T_{bit}
- Tset bit time = 8 T_{bit}

T0 = 35 + 2 * Tset + Tqui

T0 = 35 + 2 * 8 + 6

$T0 = 57\ T_{bit}$

As T0>MinTsdr: **$T1 = T0 = 57\ T_{bit}$**

As T0<MaxTsdr: **$T2 = MaxTsdr = 450\ T_{bit}$**

Use the computed values in the formula for the minimum target token rotation time:

$Ttr_{min} = n * (198 + T1 + T2) + b * 11 + 242 + T1 + T2 + Tsl$

Ttr_{min}= 5 (198+57+450)+100*11+242+57+450+600

Ttr_{min} [T_{bit}] = 5974 T_{bit}

Result:

Ttr_{min} [µs] = 5974 T_{bit} * 166.67 ns = 995.68 µs

i

Ttr is verified when it is entered into the dialog box.

If the value set for *Ttr* is lower than the value calculated by SILworX, an error message appears in the Status Viewer. A minimum value for *Ttr* is also suggested.

If *Isochronous Sync Mode* or *Isochronous Freeze Mode* is activated, the cycle time is defined by the parameter *MinSlaveInterval*. *Ttr* must be lower than *Minimum Slave Interval*.

If this condition is not met in the isochronous mode, an error message appears.

6.5 Isochronous PROFIBUS DP Cycle (DP V2 and Higher)

The PROFIBUS DP cycle consists of two telegram phases: a fixed and cyclical phase and an event-driven and acyclic phase.

The acyclic phase can extend the corresponding PROFIBUS DP cycle. This effect is not wanted in specific applications and areas, such as drive technology

To achieve a constant cycle time (t_{const}), Isochronous Mode is activated in the master such that *Min. Slave Interval [ms]* defines the constant cycle time (t_{const}). Configured in this way, the isochronous PROFIBUS DP cycle offers clock accuracy with a difference of < 10 ms.

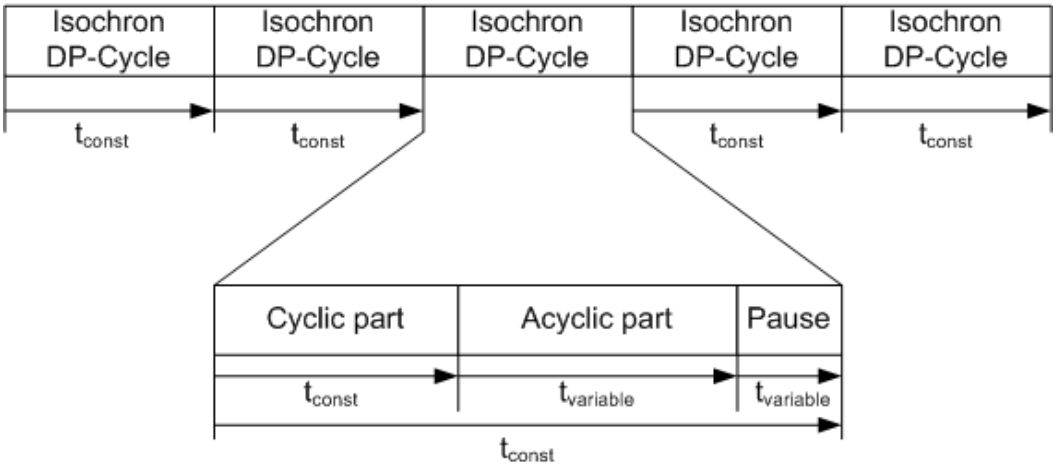


Figure 33: Isochronous PROFIBUS DP Cycle

To determine the cyclical phase, the minimum target token rotation time must be calculated.

Further, a sufficiently large time interval (typically two to three times the minimum target token rotation time *Ttr*) must be reserved for the acyclic phase. If the reserved time is not needed, a break is taken prior to starting the next cycle to ensure the cycle time remains constant. See also Chapter 6.4.3, Target Token Rotation Time (*Ttr*).

i The master is configured entering the DP cycle time determined by the user into *Min. Slave Interval [ms].*

To operate in the *Isochronous Mode*, one of the two parameters *Isochronous Sync Mode* or *Isochronous Freeze Mode* must be activated in the master.

On the bus, only one master may simultaneously operate in the Isochronous Mode. Additional masters are not permitted.

6.5.1 Isochronous Mode (DP V2 and higher)

This function allows a clock-controlled synchronization in the master and the slaves, irrespective of congestion on the bus. The bus cycle is synchronized with a clock difference of <10 ms. Highly precise positioning processes can be thus implemented.

i To a certain degree, slaves (DP V0 slaves) that do not support *Isochronous Mode* can also benefit from its advantages. To do so, the slaves must be assigned to Group 8 and the parameters *Sync* and/or *Freeze* must be activated.

The *Sync Mode* and *Freeze Mode* are normally used simultaneously.

6.5.2 Isochronous Sync Mode (DP V2 and higher)

Isochronous Sync Mode allows both a clock-controlled synchronization of the master and the slave and a simultaneous activation of the outputs of several slaves.

6.5.3 Isochronous Freeze Mode (DP V2 and higher)

Isochronous Freeze Mode allows the user to simultaneously accept the input data of several slaves.

6.6 Menu Functions of the PROFIBUS DP Slave (in the Master)

6.6.1 Creating a PROFIBUS DP Slave (in the Master)

To create a PROFIBUS DP Slave in the HIMA PROFIBUS DP Master

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master**.
2. On the context menu for PROFIBUS DP master, click **New, PROFIBUS Slave** to add a new PROFIBUS slave.

6.6.2 Edit

The Edit function from the context menu for the PROFIBUS DP master opens the **System Variables** dialog box.

The **System Variables** tab contains the following system variables that are required to evaluate the state of the PROFIBUS DP slave from within the user program.

<table>
<tr><th>Element</th><th>Description</th></tr>
<tr><td>Activation Control</td><td>A change from 0 to 1 deactivates the slave.
A change from 1 to 0 activates the slave previously deactivated.
Activated = 0
Deactivated = 1</td></tr>
<tr><td>PNO Ident Number</td><td>16 bit unique number assigned by the PNO Germany to a product (field device) and identifying it.</td></tr>
<tr><td>Standard Diagnosis</td><td>With Standard Diagnosis, the slave informs the master about its current state. This variable always contains the last received standard diagnosis. The parameters comply with the diagnostic telegram in accordance with IEC 61158.</td></tr>
<tr><td>Connection Count</td><td>It increases with each new connection. It counts from count reset.</td></tr>
<tr><td>Connection State</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>0</td><td>Deactivated:
The parameter sets are loaded for these slaves, but the slaves are completely ignored. The input data is reset to their initial values, no activity related to these slaves is noted on the bus.</td></tr>
<tr><td>1</td><td>Inactive (not connected):
If a slave can no longer be reached, the input data is reset to their initial values.
The following options can be selected for each slave:
▪ The master continues to send output data or
▪ the master attempts to re-configure the slave.</td></tr>
<tr><td>2</td><td>Active (connected):
The slaves are exchanging I/O data with the CPU.</td></tr>
</table>
</td></tr>
<tr><td>Slave Alarm Count</td><td>Number of alarms provided so far. It counts from count reset.</td></tr>
<tr><td>Standard Diagnosis Count</td><td>Number of diagnostic messages provided so far. It counts from count reset.</td></tr>
</table>

Table 80: System Variables in the PROFIBUS DP Slave

6.6.3 Properties

The **Properties** function on the context menu for the PROFIBUS DP slave opens the **Properties** dialog box. The dialog box contains the following tabs:

Tab Parameter

Element	Description
Name	Name of the slave
Address	Address of the slave Range of values: 0...125 Default value: 0
Active	Slave State Only an active slave can communicate with a PROFIBUS DP master. Default value: Activated
DP V0 Sync active	*Sync Mode* allows the user to simultaneously activate the outputs of various DP V0 slaves. **Important** This field must be deactivated in DP V2 slaves operating in *Isochronous Sync Mode*. Default value: Deactivated
DP V0 Freeze active	*Freeze Mode* allows the user to simultaneously accept the input data of several DP V0 slaves. **Important** This field must be deactivated in DP V2 slaves operating in *Isochronous Freeze Mode*. Default value: Deactivated
Watchdog Active	If the Watchdog Active checkbox is ticked, the slave detects a master's failure and enters the safe state. Default value: Deactivated
Watchdog Time [ms]	The Watchdog Active checkbox must be ticked. If master and slave do not exchange any data within this time interval, the slave disconnects itself and resets all DP output data to their initial values. 0 = Deactivated Standard value: Slave's watchdog time > 6 * Ttr Range of values: 0 ... 65535 Default value: 0
On failure send last data	FALSE: If a fault occurs, the connection is terminated and re-established. TRUE: If a fault occurs, the data continues to be sent, even without the slave's acknowledgement. Default value: Deactivated
Auto clear on failure	If *Auto clear on failure* is set to TRUE in the master and in the current slave, and if the current slave fails, the master switches the entire PROFIBUS DP into the safe state. Default value: Activated

Table 81: Parameters Tab in the PROFIBUS DP Slave

Tab Groups

In this tab, the slaves can be organized into various groups. The Global Control commands, *Sync* and *Freeze*, can systematically address one or several groups.

Element	Description	
Member of Group 1	Member of Group 1	Default value: Deactivated
Member of Group 2	Member of Group 2	
Member of Group 3	Member of Group 3	
Member of Group 4	Member of Group 4	
Member of Group 5	Member of Group 5	
Member of Group 6	Member of Group 6	
Member of Group 7	Member of Group 7	
Member of Group 8	Member of Group 8	

Table 82: Groups Tab in the Properties Dialog Box for the PROFIBUS DP Slave

Tab DP V1

This tab contains the parameters set with DP V1 and higher. In DP V0 slaves, no parameters can be selected in this tab. The Supp column shows which parameters are supported by the slave.

Element	Description
DP V1	If the DP V1 mode is not activated, no DP V1 features can be used. In this case, the slave acts like a DP V0 slave. The configuration data may have to be changed (refer to the slave manual). Default value: Deactivated
Failsafe	If this mode is activated, a master in the CLEAR state does not send zeros as output data; rather, it sends an empty data packet (failsafe data packet) to the slave. The slave recognizes that it must place the safe output data on the outputs (the value of the safe output data is not necessarily zero). Default value: Deactivated
isochronous Mode	This function allows a clock-controlled synchronization in the master and the slaves, irrespective of congestion on the bus. The bus cycle is synchronized with a clock difference of < 1 ms. Highly precise positioning processes can be thus implemented. Default value: Deactivated
Publisher Active	This function is required for the slave intercommunication. This allows the slaves to communicate with one another in a direct and time saving manner via broadcast without detouring through the master. Default value: Deactivated
Prm Block Struct. Supp.	The slave supports structured configuration data (read only). Default value: Deactivated
Check Cfg Mode	Reduced configuration control: If Check Cfg Mode is activated, the slave can operate with an incomplete configuration. This field should be deactivated during start-up. Default value: Deactivated

Table 83: DP V1 Tab in the Properties Dialog Box for the PROFIBUS DP Slave

Tab Alarms

This page is used to activate alarms. This is only possible with DP V1 slaves if DP V1 is activated and the slave supports alarms. The checkmarks in the **Supp** column designate which alarms are supported by the slave. Mandatory alarms are noted in the **Required** column.

Element	Description	
Update Alarm	Alarm, if the module parameters changed.	Default value: Deactivated
Status Alarm	Alarm, if the module state changed.	
Vendor Alarm	Vendor specific alarm.	
Diagnostic Alarm	Alarm, if specific events occur in a module, e.g., short circuits, over temperature, etc.	
Process Alarm	Alarm, if important events occur in the process.	
Pull & Plug Alarm	Alarm, if a module is removed or inserted.	

Table 84: Alarms Tab in the Properties Dialog Box for the PROFIBUS DP Slave

Tab Data

This tab specifies details about the supported data lengths and about the user data (extended configuration data).

Element	Description
Max. Input Len	Maximum length of the input data
Max. Output Len	Maximum length of the output data
Max. Data Len	Maximum total length of the input and output data
User Data Len	Length of the user data.
User Data	Configuration data. HIMA does not recommend editing at this level. Use the *User Parameters* dialog box instead, see Chapter 6.8.
Max. Diag. Data Len	Maximum length of the diagnostic data sent by the slave.

Table 85: Data Tab in the Properties Dialog Box for the PROFIBUS DP Slave

Tab Model

This tab displays self-explanatory details.

Element	Description
Model	Manufacturer identification of the PROFIBUS DP slave
Manufacturer	Manufacturer of the field device
Ident Number	Slave identification provided by PNO Germany
Revision	Issue status of the PROFIBUS DP slave
Hardware release	Hardware issue status of the PROFIBUS DP slave
Software release	Software issue status of the PROFIBUS DP slave
GSD file name	File name of the GSD file
Info Text	Additional details about the PROFIBUS DP slave

Table 86: Model Tab in the Properties Dialog Box for the PROFIBUS DP Slave

Tab Features

Element	Description
Modular Station	TRUE: Modular station FALSE: Compact station
First slot number	The modules (slots) must be numbered without gaps, starting with this value.
Max Modules	Maximum number of modules that can be installed in a modular station.
Support for 'Set Slave Add'	The slave supports dynamic address allocation.
Min. Slave Interval [ms]	The minimum time period that must elapse between two cyclic calls of the slave.
Diag. Update	Number of polling cycles until the slave's diagnosis mirrors the current state.
Support for WDBase1ms	The slave supports 1 ms as time base for the watchdogs
Support for DP V0 Sync	The slave supports DP V0 Sync
Support for DP V0 Freeze	The slave supports DP V0 Freeze
DP V1 Data Types	The slave supports the DP V1 data types.
Extra Alarm SAP	The slave supports SAP 50 for acknowledging the alarm.
Alarm Seq. Mode Count	Indicate the number of active alarms that the slave can simultaneously process. Zero means one alarm of each model.

Table 87: Features Tab in the Properties Dialog Box for the PROFIBUS DP Slave

Tab Baud Rates

This tab specifies the baud rates that the slave supports and the corresponding *MaxTsdr*.

MaxTsdr is the time within which the slave must acknowledge a request from the master. The range of values depends on the slave and the transfer rate, and ranges between 15 and 800 Tbit.

Element	Description
9.6k	MaxTsdr = 60
19.2k	MaxTsdr = 60
31.25k	Not supported
45.45k	MaxTsdr = 60
93.75k	MaxTsdr = 60
187.5k	MaxTsdr = 60
500k	MaxTsdr = 70
1.5M	MaxTsdr = 75
3M	MaxTsdr = 90
6M	MaxTsdr = 100
12M	MaxTsdr = 120

Table 88: Baud Rates Tab in the Properties Dialog Box for the PROFIBUS DP Slave

Tab Acyclic

This tab contains some parameters for the acyclic data transfer.

Element	Description
Support for C1 Read/Write	The slave supports the acyclic data transfer.
C1 Read/Write required	The slave requires the acyclic data transfer.
C1 Max Data Len[Byte]	Maximum length of an acyclic data packet.
C1 Response Timeout [ms]	Time out for the acyclic data transfer.

Table 89: Acyclic Tab in the Properties Dialog Box for the PROFIBUS DP Slave

6.7 Importing the GSD File

The GSD file contains data for configuring the PROFIBUS DP slave.

To read the GSD file for the new PROFIBUS DP slave

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, PROFIBUS Slave**.
2. On the context menu for the PROFIBUS DP master, click **Read GSD File** and select the GSD file for the PROFIBUS slave (e.g., hax100ea.gsd).

i The GSD files for HIMax controllers are available on HIMA website at www.hima.com.
The manufacturer of the field device is responsible for the correctness of the GSD file.

The GSD file *hax100ea.gsd* of the HIMax PROFIBUS DP slave provides the following modules:

PROFIBUS DP Master Input Modules	Type	Number
DP Input/ELOP Export	Byte	1
DP Input/ELOP Export	Bytes	2
DP Input/ELOP Export	Bytes	4
DP Input/ELOP Export	Bytes	8
DP Input/ELOP Export	Bytes	16
DP Input/ELOP Export	Word	1
DP Input/ELOP Export	Words	2
DP Input/ELOP Export	Words	4
DP Input/ELOP Export	Words	8
DP Input/ELOP Export	Words	16
PROFIBUS DP Master Output Modules	Type	Number
DP Output/ELOP Import	Byte	1
DP Output/ELOP Import	Bytes	2
DP Output/ELOP Import	Bytes	4
DP Output/ELOP Import	Bytes	8
DP Output/ELOP Import	Bytes	16
DP Output/ELOP Import	Word	1
DP Output/ELOP Import	Words	2
DP Output/ELOP Import	Words	4
DP Output/ELOP Import	Words	8
DP Output/ELOP Import	Words	16

Table 90: GSD File of the HIMax PROFIBUS DP Slave

6.8 Configuring User Parameters

The group's **start address** and the **number of variables** are defined in the user data field.

The number of bytes that must actually be transferred, must also be configured in the PROFIBUS DP master. This is done by choosing the PROFIBUS DP modules defined in the GDS file of the PROFIBUS DP slave (see also Chapter 6.2.2).

To open the Edit User Parameters dialog box

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master.**
2. Right-click **PROFIBUS Slave**, and then click **Properties**.
3. Select the **Data** tab and click the **Buttons ...** next to the user data.

i The structure of the **Edit User Parameters** dialog box depends on the GSD file of the slave.

Structure of the 32-byte user data field:

The 32-byte user data field is structured as follows:

The 32 bytes are allocated in **eight groups**, with **four bytes per group**.

Groups 1...4 define which and how many variables the PROFIBUS DP master receives from the PROFIBUS DP slave.

Groups 5...8 define which and how many variables the PROFIBUS DP master sends to the PROFIBUS DP slave.

The first two bytes of each group specify the start address for the first variables to be read or to be written.

The last two bytes in each group specify the number of variables that should be received or sent.

Configuring the user data in different groups:

Usually, it is not necessary to allocate variables (user data) into various groups. It is enough to define only the first signal group of the input and output variables, and to read or write the data 'en bloc'.

In applications requiring that only selected variables are read and written, up to four variable groups for both the input and the input variables can be defined.

Example

The PROFIBUS DP master sends and receives the following variables from the PROFIBUS DP slave:

1st group: **4** input variables from start address **0** and up.

2nd group: **6** input variables from start address **50** and up.

4th group: **9** input variables from start address **100** and up.

5th group: **2** output variables from start address **10** and up.

User data configuration in the PROFIBUS DP master:

Master Import/Slave Export	Start address	Number of Variables
1st group (byte 0...3)	0.0	0.4
2nd group (byte 4...7)	0.50	0.6
3rd group (byte 8...11)	0.0	0.0
4th group (byte 12...15)	0.10	0.9

Table 91: Example: Group 1...4 of the User Data Field

Master Export/Slave Import	Start Address	Number of Variable
5th group (byte 16 ... 19)	0.10	0.2
6th group (byte 20...23)	0.0	0.0
7th group (byte 24...27)	0.0	0.0
8th group (byte 28...31)	0.0	0.0

Table 92: Example: Group 1...4 of the User Data Field

Edit User Parameters dialog box of one HIMatrix or HIMax PROFIBUS DP slave.

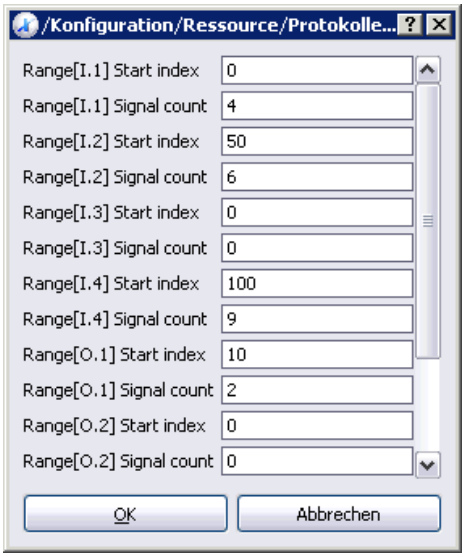


Figure 34: *Edit User Parameters* Dialog Box

6.9 PROFIBUS Function Blocks

The PROFIBUS function blocks are used to tailor the HIMA PROFIBUS DP master and the corresponding PROFIBUS DP slaves to best meet the project requirements.

The function blocks are configured in the user program such that the master and slave functions (alarms, diagnostic data, and states) can be set and read in the user program.

i Function blocks are required for special applications. They are not needed for the normal cyclic data traffic between master and slave!

For more information on the conceptual configuration of the PROFIBUS DP function blocks, refer to Chapter 12.1.

The following function blocks are available:

Function block	Function Description	Suitable beginning with Stage of Extension
MSTAT 6.9.1	Controlling the master state using the user program	DP V0
RALRM 6.9.2	Reading the alarm messages of the slaves	DP V1
RDIAG 6.9.3	Reading the diagnostic messages of the slaves	DP V0
RDREC 6.9.4	Reading the acyclic data records of the slaves	DP V1
SLACT 6.9.5	Controlling the slave states using the user program	DP V0
WRREC 6.9.6	Writing the acyclic data records of the slaves	DP V1

Table 93: Overview of the PROFIBUS DP Function Blocks

i HIMA PROFIBUS DP masters operate with the stage of extension DP V1.

HIMA PROFIBUS DP slaves operate with stage of extension DP V0.

Note that for this reason, not all function blocks of the HIMA PROFIBUS DP master can be used to control a HIMA PROFIBUS DP slaves.

6.9.1 MSTAT Function Block

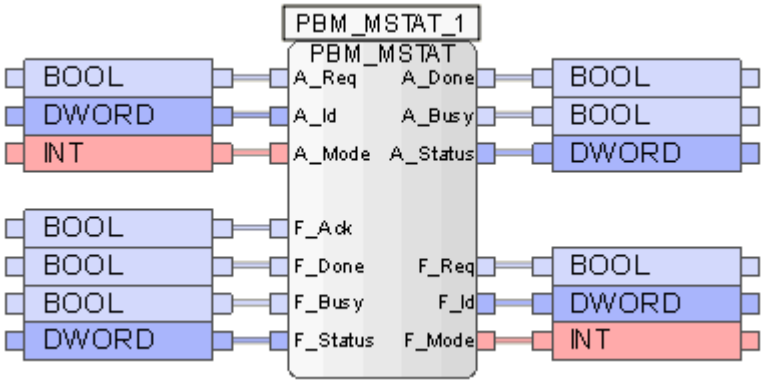


Figure 35: MSTAT Function Block

The user program uses the **MSTAT** function block (DP V0 and higher) to control the PROFIBUS DP master. The master can thus be set to one of the following states using a timer or a mechanical switch connected to a physical input.

- 0: OFFLINE
- 1: STOP
- 2: CLEAR
- 3: OPERATE

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A_Inputs	Description	Type
A_Req	Rising edge starts the function block	BOOL
A_ID	Master ID (not used)	DWORD
A_Mode	The PROFIBUS DP master can be set to the following states: 0: OFFLINE 1: STOP 2: CLEAR 3: OPERATE	INT

Table 94: A-Inputs for the MSTAT Function Block

A_Outputs	Description	Type
DONE	TRUE: The PROFIBUS DP master has been set to the state defined on the A_Mode input.	BOOL
A_Busy	TRUE: The PROFIBUS DP master is still being set.	BOOL
A_Status	Status or error code (see Chapter 6.11)	DWORD

Table 95: A-Outputs for the MSTAT Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the MSTAT function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the MSTAT function block (in the Function Blocks folder) to the MSTAT function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **MSTAT** function block in the user program to the same variables that will be connected to the outputs of the **MSTAT** function block in the structure tree.

F-Inputs	Type
F_ACK	BOOL
F_DONE	BOOL
F_BUSY	BOOL
F_STATUS	DWORD

Table 96: F-Inputs for the MSTAT Function Block

Connect the *F-Outputs* of the **MSTAT** function block in the user program to the same variables that will be connected to the inputs of the **MSTAT** function block in the structure tree.

F-Outputs	Type
F_REQ	BOOL
F_ID	DWORD
F_MODE	INT

Table 97: F-Outputs for the MSTAT Function Block

To create the MSTAT function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, Function Blocks, New.**
2. Select the **MSTAT** function block and click **OK**.
3. Right-click the **MSTAT** function block , and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **MSTAT** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **MSTAT** function block in the user program.

Inputs	Type
M_ID	DWORD
MODE	INT
REQ	BOOL

Table 98: Input System Variables

Connect the following outputs of the **MSTAT** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **MSTAT** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
DONE	BOOL
STATUS	DWORD

Table 99: Output System Variables

To use the MSTAT function block

1. In the user program, set the *A_Mode* input to the desired state.
   If *A_Mode* is not set, an error code is output after step 2 on the *A_Status* output and the PROFIBUS DP Master state is not set.
2. In the user program, set the *A_Req* input to TRUE.

---

i The function block reacts to a rising edge on *A_Req*.

☑ The *A_Busy* output is TRUE until the MSTAT command has been processed. Afterwards, *A_Busy* is set to FALSE and *A_Done* is set to TRUE.

---

i If the preset mode could not be set successfully, an error code is output on *A_Status*.

The mode of the current master can be taken from the Master Status variable (see Chapter 6.10).

6.9.2 RALRM Function Block

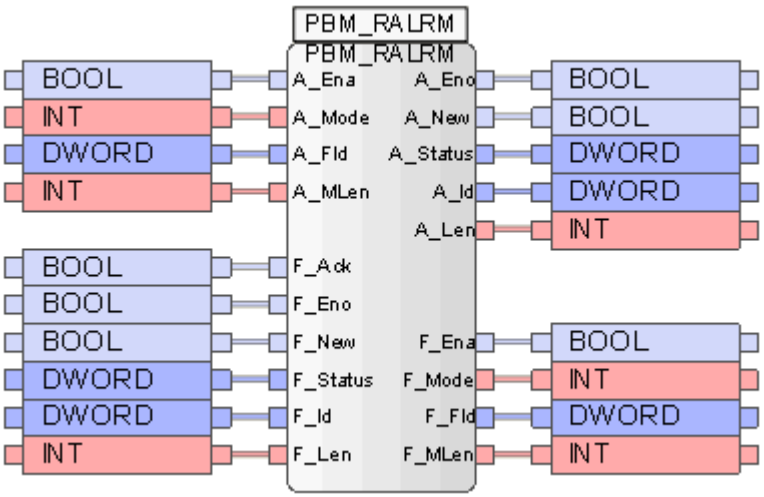


Figure 36: RALRM Function Block

The **RALRM** function block (DP V1 and higher) is used to evaluate the alarms.

Alarms are a special type of diagnostic messages that are handled with a high priority. Alarms report important events to which the application must react (e.g., a WRREC). How the application react, however, depends on the manufacturer. Refer to the manual of the PROFIBUS DP slave for more information.

As long as the **RALRM** function block is active, it waits for alarm messages from the slaves. If an alarm is received, the *A_NEW* output is set to TRUE for at least one cycle and the alarm data can be read from an alarm telegram. Before the next alarm is received, *A_NEW* is set to FALSE for at least one cycle. All alarms are acknowledged implicitly. No alarms are lost.

If several **RALRM** function blocks are used, the user program must be configured such that only one **RALRM** function block is active at any given time.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A_Inputs	Description	Type
A_Ena	TRUE enables the function block.	BOOL
A_Mode	Not used	INT
A_FID	Not used	DWORD
A_MLen	Maximum expected length of the received alarm data expressed in bytes	INT

Table 100: A-Inputs for the RDIAG Function Block

A_Outputs	Description	Type
A_Eno	TRUE: The function block is active FALSE: The function block is not active	BOOL
A_New	TRUE: New alarm was received FALSE: No new alarm	BOOL
A_Status	Status or error code (see Chapter 6.11)	DWORD
A_ID	Identification number of the slave triggering the alarm	DWORD
A_Len	Length of the received alarm data in bytes	INT

Table 101: A-Outputs for the RDIAG Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the RALRM function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **RALRM** function block (in the Function Blocks folder) to the **RALRM** function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **RALRM** function block in the user program to the same variables that will be connected to the outputs of the **RALRM** function block in the structure tree.

F-Inputs	Type
F_ACK	BOOL
F_ENO	BOOL
F_NEW	BOOL
F_STATUS	DWORD
F_ID	DWORD
F_LEN	INT

Table 102: F-Inputs for the RALRM Function Block

Connect the *F-Outputs* of the **RALRM** function block in the user program to the same variables that will be connected to the inputs of the **RALRM** function block in the structure tree.

F-Outputs	Type
F_Ena	BOOL
F_MODE	INT
F_FID	DWORD
F_MLEN	INT

Table 103: F-Outputs for the RALRM Function Block

To create the RALRM function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, Function Blocks, New.**
2. Select the **RALRM** function block and click **OK**.
3. Right-click the **RALRM** function block , and then click **Edit**..
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **RALRM** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **RALRM** function block in the user program.

Inputs	Type
EN	BOOL
F_ID	DWORD
MLEN	INT
MODE	INT

Table 104: Input System Variables

Connect the following outputs of the **RALRM** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **RALRM** function block in the user program.

Outputs	Type
ACK	BOOL
ENO	BOOL
ID	DWORD
LEN	INT
NEW	BOOL
STATUS	DWORD

Table 105: Output System Variables

The "Process Variables" tab located in the **RALRM** function block in the structure tree contains variables that must be defined and whose structure must match the alarm data. If no variables are defined, alarm data can be requested but not read.

An alarm message contains at least four bytes. The first four bytes of the alarm message contain the standard alarm data.

To simplify the evaluation of standard alarms, HIMA provides the auxiliary function block **ALARM** (see Chapter 6.10). To use it, combine the first four bytes into a variable of type DWORD and set this variable on the *IN* input of the **ALARM** auxiliary function block.

i If an alarm telegram contains more bytes than defined in the "Data" tab, only the preset number of bytes is accepted. The rest is cut off.

Alarm Data	Description
Byte 0	Length of the alarm message expressed in bytes (4...126)
Byte 1	Identification for the alarm type 1: Diagnostic alarm 2: Process alarm 3: Pull alarm 4: Plug alarm 5: Status alarm 6: Update alarm 31:Failure of a master's or a slave's extension 32...126: Manufacturer specific Consult the device manual provided by the manufacturer for more information on the specific meaning.
Byte 2	Slot number of the component triggering the alarm
Byte 3	0:No further information 1: inbound alarm, slot malfunction 2: outbound alarm, slot no longer malfunctioning 3: outbound alarm, continued slot malfunction
Byte 4 to 126	Consult the device manual provided by the manufacturer for more information on the specific meaning.

Table 106: Alarm Data

i The structure of the standard alarms (bytes 0…3) is standardized and identical for all manufacturers. Consult the manual of the PROFIBUS DP slave for more information on bytes 4…126, since their use is manufacturer specific.

Note that devices built in accordance with the DP V0 standard do not support alarm telegrams.

To use the RALRM function block

1. On the *A_Mlen* input of the user program, define the maximum amount of alarm data in bytes that must be expected. *A_Mlen* cannot be changed during operation.
2. In the user program, set the *A_Ena* input to TRUE.

i In contrast to other function blocks, the **RALRM** function block is only active as long as the *A_Ena* input is set to TRUE.

If the function block was started successfully, the *A_Eno* output is set to TRUE. If the function block could not be started, an error code is output on *A_Status*.

If a new alarm is received, the *A_New* output is set to TRUE for at least one cycle. During this time period, the alarm data of the slave triggering the alarm are contained in the outputs and can be evaluated.

Afterwards, the *A_New* output returns to FALSE for at least one cycle.
The *A_Id* and *A_Len* outputs are reset to zero before the next alarm message can be received and evaluated.

6.9.3 RDIAG Function Block

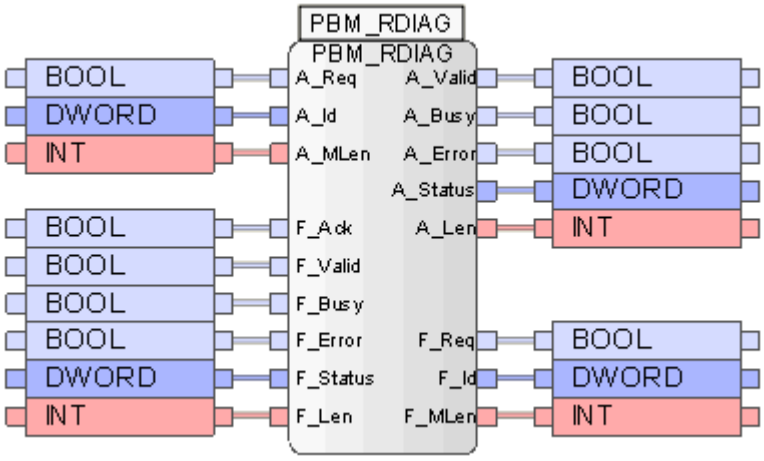


Figure 37: RDIAG Function Block

The **RDIAG** function block (DP V0 and higher) is used for reading the current diagnostic message of a slave (6...240 bytes).

As many **RDIAG** function blocks as desired may be simultaneously active within the HIMA PROFIBUS DP master.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A_Inputs	Description	Type
A_Req	The rising edge starts the reading request of a Diagnostic Message	BOOL
A_ID	Slave's identification number (see Chapter 6.10)	DWORD
A_MLen	Maximum length (in bytes) of the diagnostic message expected to be read	INT

Table 107: A-Inputs for the RDIAG Function Block

A_Outputs	Description	Type
A_Valid	A new diagnostic message has been received and is valid	BOOL
A_Busy	TRUE: Data is still being read	BOOL
ERROR	TRUE: An error occurred during the reading process	BOOL
A_Status	Status or error code (see Chapter 6.11)	DWORD
A_Len	Length of the read diagnostic data in bytes	INT

Table 108: A-Outputs for the RDIAG Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **RDIAG** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **RDIAG** function block (in the Function Blocks folder) to the **RDIAG** function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **RDIAG** function block in the user program to the same variables that will be connected to the outputs of the **RDIAG** function block in the structure tree.

F-Inputs	Type
F_ACK	BOOL
F_VALID	BOOL
F_BUSY	BOOL
F_ERROR	BOOL
F_Status	DWORD
F_LEN	INT

Table 109: F-Inputs for the RDIAG Function Block

Connect the *F-Outputs* of the **RDIAG** function block in the user program to the same variables that will be connected to the inputs of the **RDIAG** function block in the structure tree.

F-Outputs	Type
F_Req	BOOL
F_Id	DWORD
F_Mlen	INT

Table 110: F-Outputs for the RDIAG Function Block

To create the RDIAG function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, Function Blocks, New.**
2. Select the **RDIAG** function block and click **OK**..
3. Right-click the **RDIAG** function block , and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **RDIAG** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **RDIAG** function block in the user program.

Inputs	Type
ID	DWORD
MLEN	INT
REQ	BOOL

Table 111: Input System Variables

Connect the following outputs of the **RDIAG** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **RDIAG** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
ERROR	BOOL
LEN	INT
Status	DWORD
VALID	BOOL

Table 112: Output System Variables

Diagnostic Data

The **Data** tab contains variables that must be defined and whose structure must match the alarm data. A diagnostic message contains at least six bytes and a maximum of 240 bytes. The first four bytes of the diagnostic message contain the standard diagnosis.

To simplify the evaluation of the standard alarms, HIMA provides the auxiliary function block **STDDIAG** (see Chapter 6.10). To use it, combine the first four bytes into a variable of type DWORD and set this variable on the *IN* input of the **STDDIAG** auxiliary function block.

i If a diagnostic telegram contains more bytes than defined in the "Data" tab, only the preset number of bytes is accepted. The rest is cut off.

Diagnostic Data	Description
Byte 0	Byte 0...3 contains the standard diagnosis. Use the **STDDIAG** auxiliary function block to decode the standard diagnosis as a variable of the type DWORD.
Byte 1	
Byte 2	
Byte 3	Bus address of the master to which a slave is assigned.
Byte 4	High byte (manufacturer ID)
Byte 5	Low byte (manufacturer ID)
Byte 6...240	Consult the device manual provided by the manufacturer for more information on the specific meaning.

Table 113: Diagnostic Data

i The HIMA slaves send a diagnostic telegram of six bytes in length. The meaning of these bytes is standardized.

The first six bytes of slaves from other manufacturers are only functionally identical.

For more information on the diagnostic telegram, refer to the description of the slave provided by the manufacturer.

To use the RDIAG function block

1. In the user program, set the slave address on the *A_ID* input.
2. On the user program's *A_Mlen*, define the maximum amount of alarm data in bytes that must be expected.
3. In the user program, set the *A_Req* input to TRUE.

i The function block reacts to a rising edge on *A_Req*.

The *A_Busy* output is set to TRUE until the diagnostic request has been processed. Afterwards, A_Busy is set to FALSE and *A_Valid* or *A_Error* to TRUE.

If the diagnostic telegram is valid, the *A_Valid* output is set to TRUE. The diagnostic data can be evaluated using the variables defined in the "Data" tab. The *A_Len* output contains the amount of diagnostic data in bytes that was actually read.

If the diagnostic telegram could not be read successfully, the *A_Error* output is set to TRUE and an error code is output on *A_Status*.

6.9.4 RDREC Function Block

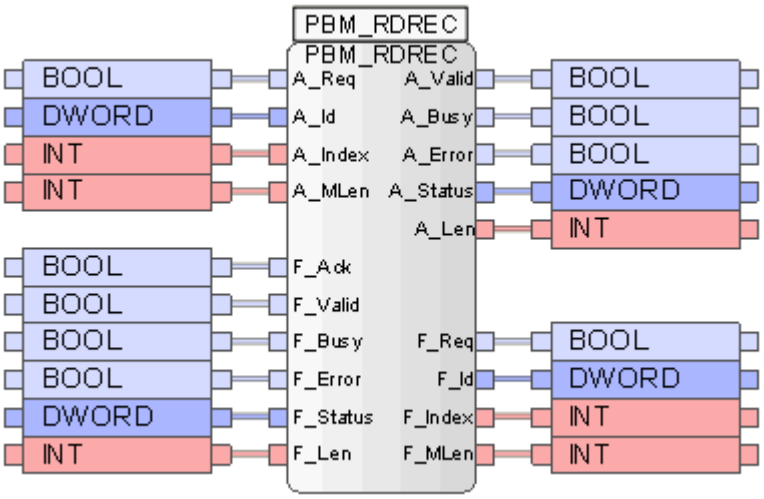


Figure 38: RDREC Function Block

The **RDREC** function block is used for acyclically reading a data record from a slave addressed on the *A_Index* input. Consult the slave's manual to find out which data can be read.

This functionality is optional and is only defined with DP V1 and higher!

Up to 32 **RDREC** and/or **WRREC** function blocks can simultaneously be active in the HIMA PROFIBUS DP Master.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A_Inputs	Description	Type
A_Req	The rising edge starts the reading request.	BOOL
A_Id	Slave identification number, (see Chapter 6.10)	DWORD
A_Index	Number of the data record to be read. Consult the device manual provided by the manufacturer for more information on the specific meaning.	INT
A_MLen	Maximum length of the data to be read in bytes.	INT

Table 114: A-Inputs for the RDREC Function Block

A_Outputs	Description	Type
A_Valid	A new data record was received and is valid.	BOOL
A_Busy	TRUE: Data is still being read.	BOOL
ERROR	TRUE: An error occurred FALSE: No error	BOOL
A_Status	Status or error code6.11, see Chapter	DWORD
A_Len	Length of the read data record information in bytes.	INT

Table 115: A-Outputs for the RDREC Function Block

Inputs and Outputs of the Function Block with Prefix F

These inputs and outputs of the function block establish the connection to the **RDREC** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **RDREC** function block (in the Function Blocks folder) to the RDREC function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **RDREC** function block in the user program to the same variables that will be connected to the outputs of the **RDREC** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Valid	BOOL
F_Busy	BOOL
F_Error	BOOL
F_Status	DWORD
F_Len	INT

Table 116: F-Inputs for the RDREC Function Block

Connect the *F-Outputs* of the **RDREC** function block in the user program to the same variables that will be connected to the inputs of the **RDREC** function block in the structure tree.

F-Outputs	Type
F_Req	BOOL
F_Id	DWORD
F_Index	INT
F_Mlen	INT

Table 117: F-Outputs for the RDREC Function Block

To create the RDREC function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, Function Blocks, New.**
2. Select the **RDREC** function block and click **OK**..
3. Right-click the **RDREC** function block , and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **RDREC** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **RDREC** function block in the user program.

Inputs	Type
ID	DWORD
INDEX	INT
MLEN	INT
REQ	BOOL

Table 118: Input System Variables

Connect the following outputs of the **RDREC** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **RDREC** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
ERROR	BOOL
LEN	INT
STATUS	DWORD
VALID	BOOL

Table 119: Output System Variables

Data	Description
No predefined variables	A user-specific data structure can be defined in the *Process Variables* tab; however, the structure must match the data record structure. For more information on the record structure, refer to the operating instructions provided by the manufacturer of the slave.

Table 120: Data

To use the RDREC function block

1. In the user program, set the slave address on the *A_ID* input.
2. In the user program, set the slave-specific index for the data record on the *A_Index* input (see the manual provided by the manufacturer).
3. In the user program, set the length of the data record to be read on the *A_Len* input.
4. In the user program, set the *A_Req* input to TRUE.

i The function block reacts to a rising edge on *A_Req*.

The *A_Busy* output is set to TRUE until the data record request has been processed. Afterwards, A_Busy is set to FALSE and *A_Valid* or *A_Error* to TRUE.

If the data record is valid, the *A_Valid* output is set to TRUE. The data set can be evaluated using the variables defined in the Data tab. The *A_Len* output contains the actual length of the data record that has been read.

If the data record could not be read successfully, the *A_Error* output is set to TRUE and an error code is output on *A_Status*.

6.9.5 SLACT Function Block

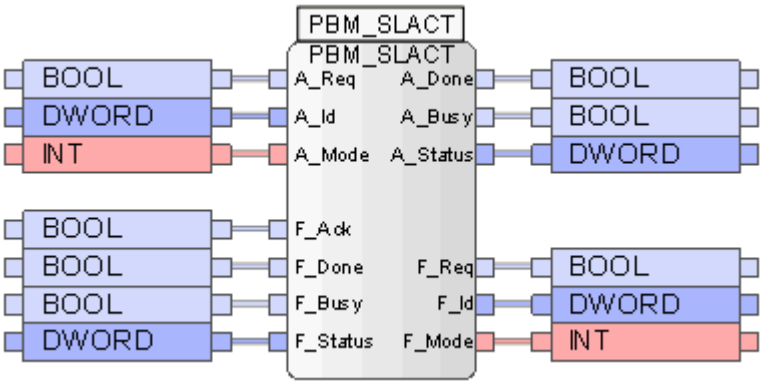


Figure 39: SLACT Function Block

The function block **SLACT** (DP V0 and higher) is used for activating and deactivating a slave from within the user program of the PROFIBUS DP master. The slave can thus be set to one of the following states using a timer or a mechanical switch connected to a physical input of the PROFIBUS DP master.

≠ 0: Active

= 0: Inactive

If various **SLACT** function blocks are used, the user program must be configured such that only one **SLACT** function block is active at a time.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A_Inputs	Description	Type
A_Req	Rising edge starts the function block	BOOL
A_Id	Slave's identification number (see Chapter 6.10)	DWORD
A_Mode	Target state for the slave PROFIBUS DP slave 0: Active (Connected) = 0: Not active (Deactivated)	INT

Table 121: A-Inputs for the SLACT Function Block

A_Outputs	Description	Type
DONE	TRUE: The PROFIBUS DP slave has been set to the state defined on the "A_Mode" input.	BOOL
A_Busy	TRUE: The PROFIBUS DP slave is still being set.	BOOL
A_Status	Status or error code (see Chapter 6.11)	DWORD

Table 122: A-Outputs for the SLACT Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **SLACT** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the SLACT function block (in the Function Blocks folder) to the SLACT function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **SLACT** function block in the user program to the same variables that will be connected to the outputs of the **SLACT** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Done	BOOL
F_Busy	BOOL
F_Status	DWORD

Table 123: F-Inputs for the SLACT Function Block

Connect the *F-Outputs* of the **SLACT** function block in the user program to the same variables that will be connected to the inputs of the **SLACT** function block in the structure tree.

F-Outputs	Type
F_Req	BOOL
F_Id	DWORD
F_Mode	INT

Table 124: F-Outputs for the SLACT Function Block

To create the SLACT function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, Function Blocks, New.**
2. Select the **SLACT** function block and click **OK**.
3. Right-click the **SLACT** function block , and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **SLACT** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **SLACT** function block in the user program.

Inputs	Type
ID	DWORD
MODE	INT
REQ	BOOL

Table 125: Input System Variables

Connect the following outputs of the **SLACT** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **SLACT** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
DONE	BOOL
STATUS	DWORD

Table 126: Output System Variables

To use the SLACT function block

1. In the user program, set the *A_Mode* input to the desired state.
2. In the user program, set the slave address identifier on the *A_ID* input.
3. In the user program, set the *A_Req* input to TRUE.

i The function block reacts to a rising edge on *A_Req*.

A_Busy output is set to TRUE until the *SLACT* command has been processed. Afterwards, *A_Busy* is set to FALSE and *A_Done* is set to TRUE.

If the slave mode could be set successfully, it is output on *A_Status*.

If the slave mode could not be set successfully, an error code is output on *A_Status*.

6.9.6 WRREC Function Block

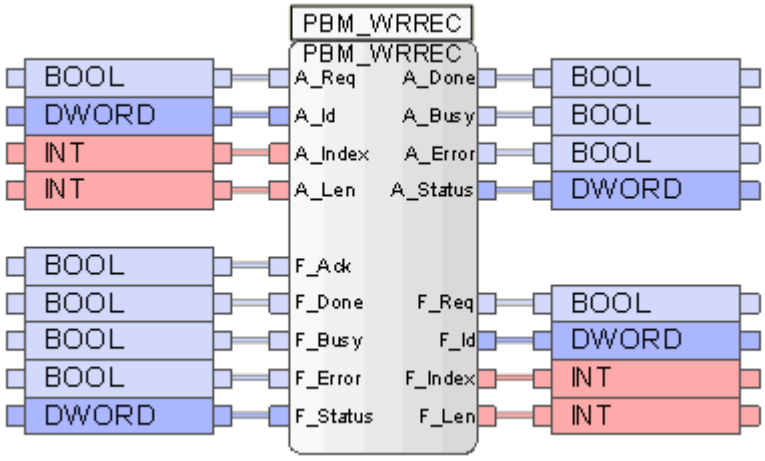


Figure 40: WRREC Function Block

The **WRREC** function block (DP V1 and higher) is used for acyclically writing a data record to a slave addressed with *A_Inde*x. Consult the slave's manual to find out which data can be written.

Up to 32 **RDREC** and/or **WRREC** function blocks can simultaneously be active in the HIMA PROFIBUS DP Master.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A_Inputs	Description	Type
A_Req	The rising edge starts the request for writing a data record.	BOOL
A_ID	Identification number of the slave (see Chapter 6.10)	DWORD
A_Index	Number of the data record to be written. Consult the device manual provided by the manufacturer for more information on the specific meaning.	INT
A_Len	Length of the data record to be written in bytes	INT

Table 127: A-Inputs for the WRREC Function Block

A_Outputs	Description	Type
DONE	TRUE: The function block completed the writing process.	BOOL
A_Busy	TRUE: The function block has not yet completed the writing process	BOOL
ERROR	TRUE: An error occurred	BOOL
A_STATUS	Status or error code (see Chapter 6.11)	DWORD

Table 128: A-Outputs for the WRREC Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **WRREC** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the WRREC function block (in the Function Blocks folder) to the WRREC function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **WRREC** function block in the user program to the same variables that will be connected to the outputs of the **WRREC** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Done	BOOL
F_Busy	BOOL
F_Error	BOOL
F_Status	DWORD

Table 129: F-Inputs for the WRREC Function Block

Connect the *F-Outputs* of the **WRREC** function block in the user program to the same variables that will be connected to the inputs of the **WRREC** function block in the structure tree.

F-Outputs	Type
F_Req	BOOL
F_Id	DWORD
F_Index	INT
F_Len	INT

Table 130: F-Outputs for the WRREC Function Block

To create the WRREC function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master, Function Blocks, New.**
2. Select the **WRREC** function block and click **OK**.
3. Right-click the **WRREC** function block , and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **WRREC** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **WRREC** function block in the user program.

Inputs	Type
ID	DWORD
INDEX	INT
LEN	INT
REQ	BOOL

Table 131: Input System Variables

Connect the following outputs of the **WRREC** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **WRREC** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
DONE	BOOL
ERROR	BOOL
STATUS	DWORD

Table 132: Output System Variables

Data	Description
No predefined variables	A user-specific data structure can be defined in the *Process Variables* tab; however, the structure must match the data record structure. For more information on the record structure, refer to the operating instructions provided by the manufacturer of the slave.

Table 133: Data

To operate the function block WRREC, the following steps are essential:

1. In the user program, set the slave address on the *A_ID* input.
2. In the user program, set the slave-specific index for the data record on the *A_Index* input (see the manual provided by the manufacturer).
3. In the user program, set the length of the data record to be written on the *A_Len* input.
4. In the user program, set the data record as defined in the "Data" tab.
5. In the user program, set the *A_Req* input to TRUE.

i The function block reacts to a rising edge on *A_Req*.

The *A_Busy* output is TRUE until to the data record is written. Afterwards, *A_Busy* is set to FALSE and *A_Done* is set to TRUE.

If the data record could not be written successfully, the *A_Error* output is set to TRUE and an error code is output on *A_Status*.

6.10 PROFIBUS Auxiliary Function Blocks

The auxiliary function blocks are used to configure and evaluate the inputs and outputs of the function blocks.

The following auxiliary function blocks are available:

Auxiliary Function Blocks	Function Description
ACTIVE (see Chapter 6.10.1)	Determine if the slave is active or inactive
ALARM (see Chapter 6.10.2)	Decode the alarm data
DEID (see Chapter 6.10.3)	Decode the identification number
ID (see Chapter 6.10.4)	Generate a four byte identifier
NSLOT (see Chapter 6.10.5)	Create a continuous identification number for the slots
SLOT (see Chapter 6.10.6)	Create a SLOT identification number using a slot number
STDDIAG (see Chapter 6.10.7)	Decode the standard diagnosis of a slave
LATCH	Only used within other function blocks
PIG	Only used within other function blocks
PIGII	Only used within other function blocks

Table 134: Overview of the Auxiliary Function Blocks

6.10.1 ACTIVE Auxiliary Function Block

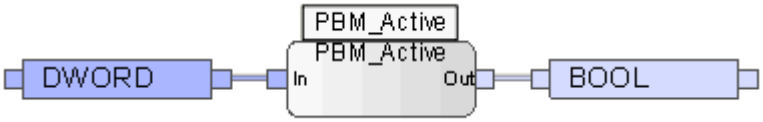


Figure 41: ACTIVE Auxiliary Function Block

The ACTIVE auxiliary function block uses the standard diagnosis of a PROFIBUS DP slave to determine if the slave is active or inactive.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs	Description	Type
IN	Slave standard diagnosis	DWORD

Table 135: Inputs for the ACTIVE Auxiliary Function Block

Outputs	Description	Type
OUT	TRUE: The slave is active FALSE: The slave is inactive	BOOL

Table 136: Outputs for the ACTIVE Auxiliary Function Block

6.10.2 Auxiliary Function Block ALARM

(Decode the Alarm Data)

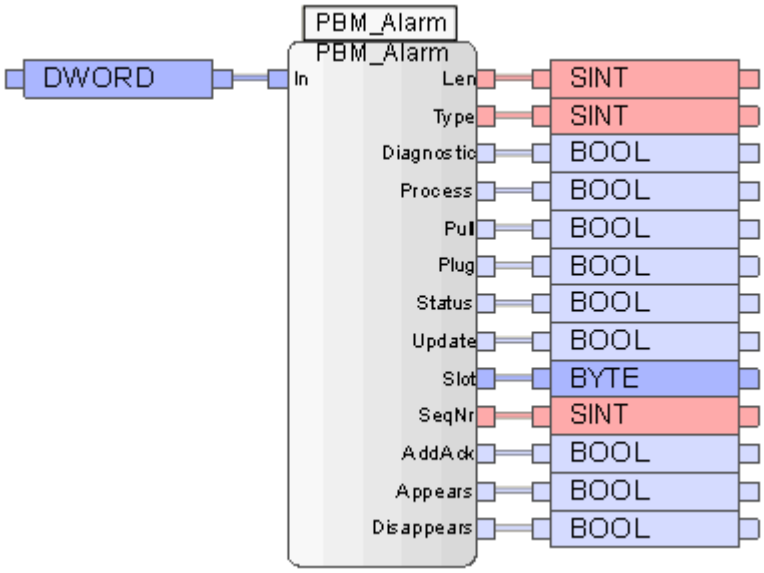


Figure 42: ALARM Auxiliary Function Block

The **ALARM** auxiliary function block decodes the standard alarm data of a PROFIBUS DP slave.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs	Description	Type
IN	Standard alarm	DWORD

Table 137: Inputs for the ALARM Auxiliary Function Block

Output	Description	Type
Len	Total length of the alarm message.	SINT
Type	1: Diagnostic alarm 2: Process alarm 3: Pull alarm 4: Plug alarm 5: Status alarm 6: Update alarm The other numbers are either reserved or manufacturer specific. Consult the device manual provided by the manufacturer for more information on the specific meaning.	SINT
Diagnostic	True = Diagnostics alarm	BOOL
Process	True = Process alarm	BOOL
Pull	True = The module was pulled	BOOL
Plug	True = The module was plugged	BOOL
Status	True = Status alarm	BOOL
Update	True = Update alarm	BOOL
Slot	Alarm Triggering Module	BYTE
SeqNr	Alarm Sequence Number	SINT

<table>
<tr><th>Output</th><th>Description</th><th>Type</th></tr>
<tr><td>AddAck</td><td>TRUE means that the slave that triggered this alarm requires an additional acknowledgement from the application. For more information on this, consult the slave manual provided by the manufacturer.</td><td>BOOL</td></tr>
<tr><td>Appears
Disappears</td><td>
<table>
<tr><th>Output</th><th>Value</th><th>Description</th></tr>
<tr><td>Appears</td><td>TRUE</td><td rowspan="2">If both are FALSE, no error has occurred yet.</td></tr>
<tr><td>Disappears</td><td>FALSE</td></tr>
<tr><td>Appears</td><td>TRUE</td><td rowspan="2">An error occurred and is still present.</td></tr>
<tr><td>Disappears</td><td>FALSE</td></tr>
<tr><td>Appears</td><td>TRUE</td><td rowspan="2">An error occurred and is disappearing.</td></tr>
<tr><td>Disappears</td><td>FALSE</td></tr>
<tr><td>Appears</td><td>TRUE</td><td rowspan="2">If both are TRUE, the error disappears but the slave remains in a malfunction state.</td></tr>
<tr><td>Disappears</td><td>FALSE</td></tr>
</table>
</td><td>BOOL</td></tr>
</table>

Table 138: Outputs for the ALARM Auxiliary Function Block

6.10.3 DEID Auxiliary Function Block

(Decode the identification number)

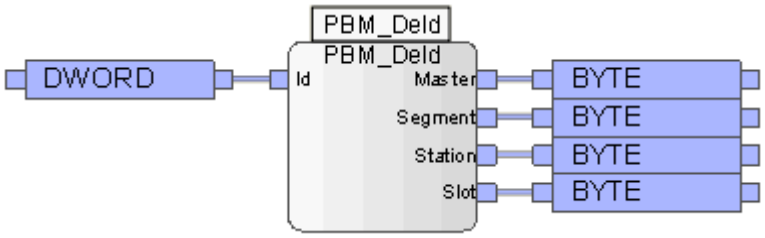


Figure 43: DEID Auxiliary Function Block

The **DEID** auxiliary function block decodes the identification number and disjoints it into its four component parts.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs	Description	Type
ID	Identification number of the slave	DWORD

Table 139: Inputs for the DEID Auxiliary Function Block

Outputs	Description	Type
Master	Master bus address	BYTE
Segment	Segment	BYTE
Stop	Slave bus address	BYTE
Slot	Slot or module number	BYTE

Table 140: Outputs for the DEID Auxiliary Function Block

6.10.4 ID Auxiliary Function Block

(Generate the identification number)

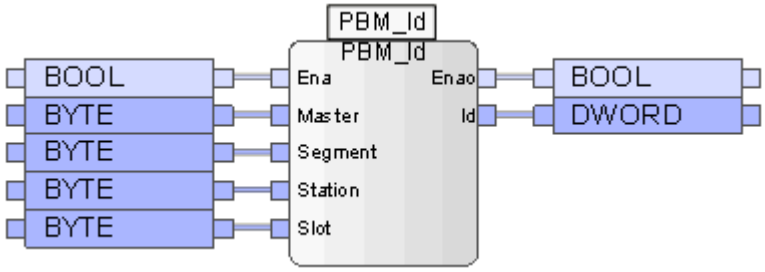


Figure 44: ID Auxiliary Function Block

The **ID** auxiliary function block uses four bytes to generate an identifier (identification number) used by other auxiliary function blocks.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs	Description	Type
Ena	Not used	BOOL
Master	Bus address	BYTE
Segment	Segment	BYTE
Stop	Slave bus address	BYTE
Slot	Slot or module number	BYTE

Table 141: Inputs for the ID Auxiliary Function Block

Outputs	Description	Type
Enao	Not used	BOOL
ID	Identification number of the slave	DWORD

Table 142: Outputs for the ID Auxiliary Function Block

6.10.5 NSLOT Auxiliary Function Block

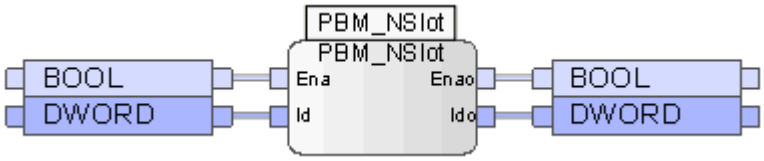


Figure 45: NSLOT Auxiliary Function Block

The **NSLOT** auxiliary function block uses an identifier to generate a new identifier that addresses the next slot within the same slave. Ena must be set to TRUE to allow the auxiliary function block to run.

Enao is set to TRUE if the result on the Ido output is valid.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs	Description	Type
Ena	The auxiliary function block runs as long as Ena is set to TRUE.	BOOL
ID	Identification number of the slave	DWORD

Table 143: Inputs for the NSLOT Auxiliary Function Block

Outputs	Description	Type
Enao	TRUE = The result is valid FALSE = No further slot number	BOOL
Ido	Identification number of the slave	DWORD

Table 144: Outputs for the NSLOT Auxiliary Function Block

6.10.6 SLOT Auxiliary Function Block

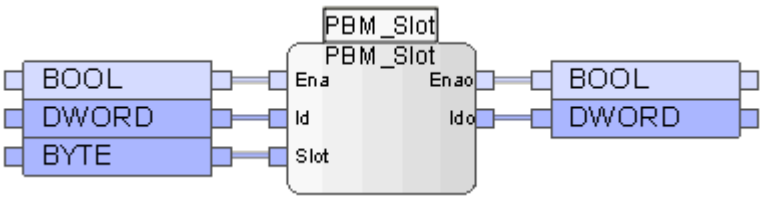


Figure 46: SLOT Auxiliary Function Block

The **SLOT** auxiliary function block uses an identifier and a slot number to generate a new identifier that addresses the same slave as the first identifier but with the new slot number.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs	Description	Type
Ena	Not used	BOOL
ID	Logical address of the slave component (slave ID and slot number)	DWORD
Slot	New slot or module number	BYTE

Table 145: Inputs for the SLOT Auxiliary Function Block

Outputs	Description	Type
Enao	Not used	BOOL
Ido	Identification number of the slave	DWORD

Table 146: Outputs for the SLOT Auxiliary Function Block

6.10.7 STDDIAG Auxiliary Function Block

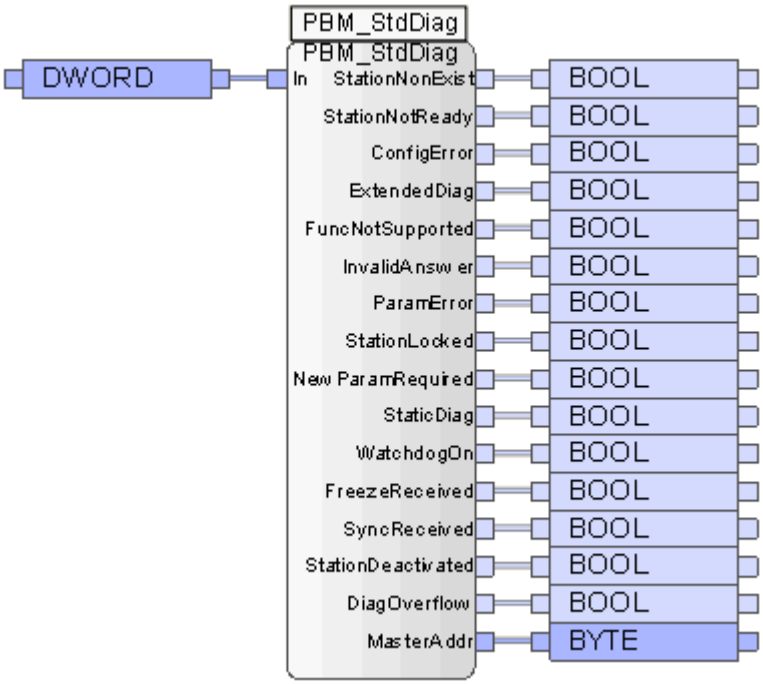


Figure 47: STDDIAG Auxiliary Function Block

The **STDDIAG** auxiliary function block decodes the standard diagnosis of a PROFIBUS DP Slave.

The outputs of type BOOL in the **STDDIAG** auxiliary function block are set to TRUE if the corresponding bit has been set in the standard diagnosis.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs	Description	Type
IN	Slave standard diagnosis	DWORD

Table 147: Inputs for the STDDIAG Auxiliary Function Block

Outputs	Description	Type
StationNonExist	The slave does not exist	BOOL
StationNotReady	Slave not ready	BOOL
ConfigError	Configuration error	BOOL
ExtendedDiag	Extended diagnosis follows	BOOL
FuncNotSupported	The function is not supported	BOOL
InvalidAnswer	Invalid reply from slave	BOOL
ParamError	Parameter error	BOOL
StationLocked	Slave locked by another master	BOOL
NewParamRequired	New configuration data required	BOOL
StaticDiag	Static diagnosis	BOOL
WatchdogOn	Watchdog active	BOOL
FreezeReceived	Freeze command received	BOOL
SyncReceived	Sync command received	BOOL
StationDeactivated	The slave was deactivated	BOOL
DiagOverflow	Diagnostics overflow	BOOL
MasterAddr	Master bus address	BYTE

Table 148: Outputs for the STDDIAG Auxiliary Function Block

To read the standard diagnosis of the PROFIBUS DP slave:

1. In the structure tree, open **Configuration, Resource, Protocols, PROFIBUS DP Master.**
2. Right-click **PROFIBUS Slave** , and then click **Edit**.
3. Drag the global variable of type DWORD onto the *Standard Diagnosis* field.
4. Connect this global variable with the input of the **STDDIAG** function block.

6.11 Error Codes of the Function Blocks

If a function block is unable to correctly execute a command, an error code is output on **A_Status**. The meaning of the error codes is described in the following table.

Error Code	Symbol	Description
16#40800800	TEMP_NOT_AVAIL	Service temporary not available
16#40801000	INVALID_PARA	Invalid parameter
16#40801100	WRONG_STATE	The slave does not support DP V1
16#40808000	FATAL_ERR	Fatal program error
16#40808100	BAD_CONFIG	Configuration error in the data area
16#40808200	PLC_STOPPED	The controller was stopped
16#4080A000	READ_ERR	Error while reading a record
16#4080A100	WRITE_ERR	Error while writing a record
16#4080A200	MODULE_FAILURE	The error cannot be specified in greater detail
16#4080B000	INVALID_INDEX	Index is invalid
16#4080B100	WRITE_LENGTH	Wrong length while writing
16#4080B200	INVALID_SLOT	Slot number invalid
16#4080B300	TYPE_CONFLICT	Wrong type
16#4080B400	INVALID_AREA	Wrong read/write range
16#4080B500	STATE_CONFLICT	Master in the wrong state
16#4080B600	ACCESS_DENIED	Slave not active (or similar)
16#4080B700	INVALID_RANGE	Wrong read/write range
16#4080B800	INVALID_PARAMETER	Wrong parameter value
16#4080B900	INVALID_TYPE	Wrong parameter type
16#4080C300	NO_RESOURCE	Slave not available
16#4080BA00	BAD_VALUE	Invalid value
16#4080BB00	BUS_ERROR	Bus Error
16#4080BC00	INVALID_SLAVE	Invalid slave ID
16#4080BD00	TIMEOUT	Timeout occurred
16#4080C000	READ_CONSTRAIN	Read constraint
16#4080C100	WRITE_CONSTRAIN	Write constraint
16#4080C200	BUSY	A function block of this type is already active
16#4080C300	NO_RESOURCE	Slave inactive

Table 149: Error Codes of the Function Blocks

6.12 Control Panel (PROFIBUS DP Master)

The Control Panel can be used to verify and control the settings for the PROFIBUS DP master. Details about the current status of the master or slave associated with it (e.g., cycle time, bus state, etc.) are displayed.

To open Control Panel for monitoring the PROFIBUS DP master

1. In the structure tree, click **Resource**.
2. Click **Online** on the **Action Bar**.
3. In the **System Log-in** window, enter the access data to open the Control Panel for the resource.
4. In the structure tree associated with the Control Panel, select **PROFIBUS DP Master**.

6.12.1 Context Menu (PROFIBUS DP Master)

The following commands can be chosen from the context menu for the selected PROFIBUS DP master:

Offline:

Switch off the selected PROFIBUS DP master. If the master is switched off, it cannot perform any actions.

Stop:

Stop the selected PROFIBUS DP master. The master participates in the token protocol but does not send any data to the slaves.

Clear:

By clicking the CLEAR button, the selected PROFIBUS DP master adopts a safe state and exchanges safe data with the slaves. The output data sent to the slaves only contains zeros. Failsafe slaves receive failsafe telegrams containing no data. The PROFIBUS DP master ignores the input data from the slaves, and uses the initial values in the user program instead.

Operate:

Start the selected PROFIBUS DP master. The PROFIBUS DP master cyclically exchanges I/O data with the slaves.

Reset Statistics:

The Reset Statistical Data button resets the statistical data (cycle [min], cycle [max] …) to zero.

6.12.2 View Box (PROFIBUS Master)

The view box displays the following values of the selected PROFIBUS DP master.

Element	Description
Name	Name of the PROFIBUS DP Master
Baud rate [bps]	Baud rate of the master The master can communicate using all baud rates specified in the standard. Cycle times can be set up to a lower limit of 2 ms.
Fieldbus interface	FB1, FB2
Fieldbus address	Master bus address (0 ... 125)
Master State	Indicate the current protocol state (see Chapter 6.12.3). 0 = OFFLINE 1 = STOP 2 = CLEAR 3 = OPERATE 100 = UNDEFINED
Bus state	Bus error code 0...6: 0 = OK 1 = Address error: The master address is already available on the bus 2 = Bus malfunction: A malfunction was detected on the bus, e.g., bus was not properly terminated, and several stations are sending data simultaneously. 3 = Protocol error: An incorrectly coded packet was received. 4 = Hardware fault: The hardware reported a fault, e.g., too short time periods. 5 = Unknown error: The master changed the state for an unknown reason. 6 = Controller Reset: With severe bus malfunctions, the controller chip could adopt an undefined state and is reset. The error code retains its value until the bus error has been eliminated.
Bus Error Count	Number of the bus error, so far.
CPU Load (planned) [%]	Load of the COM module planned for this protocol.
CPU Load (actual) [%]	Actual load of the COM module for this protocol.

Element	Description
MSI [ms]	Min. Slave Interval in ms, resolution 0.1 ms
TTR [ms]	Target Token Rotation Time in ms, resolution 0.1 ms
Last Cycle Time [ms]	Last PROFIBUS DP cycle time [ms]
Minimum Cycle Time [ms]	Minimum PROFIBUS DP cycle time [ms]
Average Cycle Time [ms]	Average PROFIBUS DP cycle time [ms]
Maximum Cycle Time [ms]	Maximum PROFIBUS DP cycle time [ms]

Table 150: View Box of the PROFIBUS Master

6.12.3 PROFIBUS DP Master State

The master status is displayed in the view box of the Control Panel and can be evaluated in the user program using the Master Connection State status variable.

Master State	Master State
OFFLINE	The master is switched off; no bus activity.
STOP	The master participates in the token protocol but does not send any data to the slaves.
CLEAR	The master is in the safe state and exchanges data with the slaves. ▪ The output data sent to the slaves only contains zeros. ▪ Failsafe slaves receive failsafe telegrams containing no data. ▪ The input data of the slaves are ignored and instead initial values are used.
OPERATE	The master is operating and cyclically exchanges I/O data with the slaves.
UNDEFINED	The firmware for the PROFIBUS DP master module is being updated.

Table 151: PROFIBUS DP Master State

6.12.4 Behavior of the PROFIBUS DP Master

Behavior of the PROFIBUS DP master according to the controller operating state.

State of the Controller	Behavior of the HIMA PROFIBUS DP Master
STOP *)	If the controller is in STOP, the master is in the OFFLINE state.
RUN	If the controller is in RUN, the master tries to enter the OPERATE state.
STOP	If the controller enters the STOP state, the master adopts the CLEAR state. If the master is already in the STOP or OFFLINE state, it remains in this state.

*) After powering up the controller or loading the configuration

Table 152: Behavior of the PROFIBUS DP Master

6.12.5 Function of the FBx LED in the PROFIBUS Master

The state of the serial PROFIBUS DP communication is displayed with the FBx LED on the corresponding configured serial interfaces (fb1, fb2).

FBx LED	Description
OFF	No configuration or invalid configuration of the PROFIBUS DP master.
Blinking Every 2 seconds	Valid configuration. The PROFIBUS DP master is in the OFFLINE or STOP state.
ON	The PROFIBUS DP master is in the OPERATE or CLEAR state and exchanges data with all the activated slaves.
Blinking, every second	At least one slave failed.

Table 153: FBx LED (PROFIBUS DP Slave)

6.13 HIMA PROFIBUS DP Slave

This chapter describes the characteristics of the HIMA PROFIBUS DP slave and the menu functions and dialog boxes in SILworX required for configuring the HIMA PROFIBUS DP slave.

Equipment and System Requirements

Element	Description
HIMA controller	HIMax with COM module
COM module	The serial fieldbus interface (FB1 or FB2) used on the HIMax COM module must be equipped with an optional HIMA PROFIBUS DP slave submodule. For more information on the interface assignment, see Chapter 3.7.
Activation	Activation through the plug-in module, see Chapter 3.5.

Table 154: Equipment and System Requirements for the HIMA PROFIBUS DP Slave

PROFIBUS DP Slave Properties

Element	Description
Type of HIMA PROFIBUS DP slave	DP V0
Transfer rate	9.6 kbit/s ... 12 Mbit/s
Bus address	0 ...125
Max. number of slaves	One HIMA PROFIBUS DP slave can be configured for each COM module.
Process data volume of a HIMA PROFIBUS DP slave	DP-Output: max. 192 bytes DP-Input: max. 240 bytes Total: max. 256 bytes
Protocol watchdog	If the COM is in RUN and the connection to the PROFIBUS DP master is lost, the DP slave detects this once the watchdog timeout has expired (the watchdog timeout must be set in the master). In this case, the DP output data (or input data from the perspective of the resource) are reset to their initial value and the *Data Valid* flag (status variable of the DP slave protocol) is set to FALSE.

Table 155: Properties of the HIMA PROFIBUS DP Slave

6.13.1 Creating a HIMA PROFIBUS DP Slave

To create a new HIMA PROFIBUS DP Slave

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, click **New, PROFIBUS DP Slave** to add a new PROFIBUS DP slave.
3. On the context menu for the PROFIBUS DP slave, click **Edit**.
4. In the **Properties** tab, click **Module** and **Interface**.

6.14 Menu Functions of the PROFIBUS DP Slave

6.14.1 Edit

The **Edit** dialog box for the PROFIBUS DP master contains the following tabs:

Process Variables

The send and receive variables are created in the **Process Variables** tab.

Input Variables

The variables that the current controller should receive are entered in the *Input Signals* area.

Any variables can be created in the *Input Signals* area. Offsets and types of the received variables must be identical with offsets and types of the send variables of the communication partner.

Output Variables

The variables for cyclic data exchange sent by this controller are entered in the *Output Signals* area.

Any variables can be created in the *Output Signals* area. Offsets and types of the received variables must be identical with offsets and types of the receive variables of the communication partner.

System Variables

The variables that should be read in the controller are defined in the **System Variables** tab.

The **System Variables** tab contains the following system variables that are required to evaluate the state of the PROFIBUS DP slave from within the user program.

Element	Description
Current baud rate	Baud rate currently used by the PROFIBUS DP slave protocol.
Data valid	If the status variable *Data Valid* is set to TRUE, the slave received valid import data from the master. The status variable is set to FALSE if the watchdog time within the slave has expired. Default value: FALSE Note: If the master did not activate the slave's watchdog and the connection is lost, the *Data Valid* status variable retains the value TRUE since the PROFIBUS DP slave has no means to recognize that the connection was lost. This fact must be taken into account when using this variable!
Error Code	If an error occurred in the PROFIBUS DP slave protocol, the error is transferred to this variable. The last occurred error is displayed. Possible (hexadecimal) value: 0x00: No error 0xE1: faulty configuration by the PROFIBUS Master 0xD2: faulty configuration by the PROFIBUS DP Master Default value: 0x00
Master ID	This is the ID of the PROFIBUS master that configured its own PROFIBUS DP slave. Possible (decimal) values: 0-125: Master ID 255: The slave is not assigned to any master Default value: 0xFF
Protocol State	Describe the state of the PROFIBUS DP slave protocol Possible (hexadecimal) value: 0xE1: The controller is disconnected from the bus or not active. 0xD2: The controller waits for a configuration from the master. 0xC3: The controller cyclically exchanges data with the master. Default value: 0xE1
Slave ID	This variable contains the controller's PROFIBUS DP slave ID used on the bus. The user used the PADT to configure the slave ID. Possible (decimal) values: 0-125: PROFIBUS DP Slave ID of the own controller Default value: 0xFF
Watchdog Time	Watchdog time in milliseconds configured in the master. See Chapter 6.6.3.

Table 156: System Variables in the PROFIBUS DP Slave

6.14.2 Properties

The **Properties** tab for the HIMA PROFIBUS DP slave contains the following parameters for configuring the PROFIBUS DP slave.

The value of the default parameters *In one cycle* and *Refresh Rate [ms]* provide a fast means of exchanging PROFIBUS DP data between the COM module (COM) and the PROFIBUS DP slave hardware of the HIMax controller.

These parameters should only be changed if it is necessary to reduce the COM load for an application, and the process allows this change.

i Only experienced programmers should modify the parameters.
Increasing the refresh rate for the COM and PROFIBUS DP hardware means that the effective refresh rate of the PROFIBUS DP data is also increased. The system time requirements must be verified.

Also take the parameter **Min. Slave Interval [ms]** into account (see Timings Tab, Chapter 6.3.2) which defines the minimum refresh rate of the PROFIBUS DP data between PROFIBUS DP master and PROFIBUS DP slave.

Element	Description
Type	PROFIBUS DP slave
Name	Name of the PROFIBUS DP Slave
Within one cycle	Activated: Transfer of all protocol data from the CPU to the COM within a CPU cycle. Deactivated: Transfer of all protocol data from the CPU to the COM, distributed over multiple CPU cycles, each with 1100 byte per data direction. This can also allow lowering the cycle time of the controller. Default value: Activated
Module	Selection of the COM module within which the protocol is processed.
Use Max CPU Load	Activated: Use CPU load limit from the Max. CPU Load [%] field. Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%

<table>
<tr><td>Station address</td><td>Slave station address.
Only one slave station address may be available on the bus.
Range of values: 1 ... 125
Default value: 0</td></tr>
<tr><td>Refresh Rate [ms]</td><td>Refresh rate in milliseconds at which the COM and the PROFIBUS DP slave hardware exchange protocol data.
Range of values: 4...1000
Default value: 10</td></tr>
<tr><td>Interface</td><td>Fieldbus interface that should be used for the PROFIBUS DP slave.
Range of values: fb1, fb2
Default value: None</td></tr>
<tr><td>Baud rate [bps]</td><td>Baud rate used for the bus.
Possible values:
<table>
<tr><th>Value</th><th>Baud Rate</th><th>FB1</th><th>FB2</th></tr>
<tr><td>9600</td><td>9.6 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>19200</td><td>16.2 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>45450</td><td>45.45 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>93750</td><td>93.75 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>187500</td><td>187.5 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>500000</td><td>500 kbit/s</td><td>X</td><td>X</td></tr>
<tr><td>1500000</td><td>1.5 Mbit/s</td><td>X</td><td>X</td></tr>
<tr><td>3000000</td><td>3 Mbit/s</td><td>X</td><td>-</td></tr>
<tr><td>6000000</td><td>6 Mbit/s</td><td>X</td><td>-</td></tr>
<tr><td>12000000</td><td>12 Mbit/s</td><td>X</td><td>-</td></tr>
</table></td></tr>
</table>

Table 157: Slave Properties: General Tab

6.15 Control Panel (Profibus DP Slave)

The Control Panel can be used to verify and control the settings for the PROFIBUS DP slave. Details about the slave's current status (e.g., cycle time, bus state, etc.) are displayed.

To open Control Panel for monitoring the PROFIBUS DP Slave

1. In the structure tree, click **Resource**.
2. Click **Online** on the **Action Bar**.
3. In the **System Log-in** window, enter the access data to open the Control Panel for the resource.
4. In the structure tree associated with the Control Panel, select **PROFIBUS DP Slave**.

6.15.1 Context Menu (PROFIBUS DP Slave)

The following commands can be chosen from the context menu for the selected PROFIBUS DP slave:

Activate:

Activate the selected slave which can now exchange data with the PROFIBUS DP master.

Deactivate:

Deactivate the selected slave. The communication is terminated.

6.15.2 View Box (PROFIBUS DP Slave)

The view box displays the following values of the selected PROFIBUS DP master.

Element	Description
Name	Name of the PROFIBUS DP Slave
Fieldbus interface	Assigned fieldbus interface of the slave
Protocol State	Connection State 0 = Deactivated, 1 = Inactive (connection attempt) 2 = Connected
Error State	See Chapter 6.14.1
Timeout	Watchdog time in milliseconds configured in the master. See Chapter 6.6.3
Watchdog Time [ms]	It is set in the master. See Chapter 6.6.3.
Fieldbus address	See Chapter 6.14.2.
Master Address	Address of the PROFIBUS DP master.
Baud rate [bps]	Current baud rate. See Chapter 6.14.2.
CPU Load (planned) [%]	Load of the COM module planned for this protocol.
CPU Load (actual) [%]	Actual load of the COM module for this protocol.

Table 158: View Box of the PROFIBUS DP Slave

6.16 Function of the FBx LED in the PROFIBUS Slave

The COM module indicates the state of the local PROFIBUS DP slave protocol using one of the LEDs assigned to the fieldbus interface. The states of these LEDs are specified in the following table.

FBx LED	Color	Description
OFF	Yellow	The PROFIBUS DP slave protocol is not active! I.e., the controller is in the STOP state or no PROFIBUS DP slave is configured.
Blinking every 2 seconds	Yellow	No data traffic! The PROFIBUS DP slave is configured and ready.
ON	Yellow	The PROFIBUS DP slave protocol is active and is exchanging data with the PROFIBUS DP master.
OFF	Red	PROFIBUS DP Slave protocols not disturbed.
Blinking	Red	The following events result in a malfunction. The configurations of the PROFIBUS DP master and slave are faulty or do not correspond to one another. Calculating time budget exceeded If no faults occur for a period longer than 5 seconds, the state changes to "Protocol not disturbed".

Table 159: LED FBx (PROFIBUS DP Slave)

7 Modbus

The Modbus coupling of HIMax systems to almost any process control and visual display system can be achieved either directly, using the RS485 interfaces, or indirectly, using the Ethernet interfaces of the controllers. The HIMax system can be operated as a master or as a slave.

The Modbus functionality makes it particularly easy to connect to Control Panels or other controllers. Given its intensive use in projects worldwide, Modbus has been proven through countless applications.

Modbus master (see Chapter 7.1)
Redundancy of the Modbus master must be configured in the user program such that the user program monitors the redundant transmission paths and assigns the redundantly transmitted process data to the corresponding transmission path.

Modbus slave (see Chapter 7.5.3)
The Modbus slave can be configured redundantly.

i HIMax controllers and the communication partner must be located in the same subnet, if the Ethernet interfaces are used as transmission channel, or they must have the corresponding routing settings if a router is used.

7.1 HIMA Modbus Master

Both, the serial interface (RS485) and TCP/UDP (Ethernet) can be used to configure the data transfer between the HIMA Modbus master and the Modbus slaves. Further, the HIMA Modbus master can also be used as a gateway (Modbus: TCP/UDP -> RS485).

Equipment and System Requirements

Element	Description
HIMA controller	HIMax with COM module
Processor module	The Ethernet interfaces on the processor module may not be used for Modbus TCP.
COM module	Ethernet 10/100BaseT Pin assignment of the D-sub connectors FB1 and FB2 If Modbus RTU is used, the serial fieldbus interface (FB1 or FB2) used on the COM module must be equipped with an optional HIMA RS485 submodule. For more information on the interface assignment, see Chapter 3.7.
Activation	Each of the two Modbus master functions must be enabled individually, see Chapter 3.5. Modbus Master RTU (RS485), Modbus Master TCP

Table 160: Equipment and System Requirements for the Modbus Master

Modbus Master Properties

Property	Description
Modbus master	One Modbus master can be configured for each COM module. One Modbus master can simultaneously: operate TCP/UDP slaves and serial slaves on several serial buses; be used as gateway from Modbus TCP to Modbus RTU.
Max. number of Modbus slaves	One Modbus master can operate up to 247 slaves. 121 Modbus slaves per serial interface 64 TCP slaves via TCP/IP connection 247 UDP slaves through the UDP/IP connection
Max. number of request telegrams	Up to 988 request telegrams can be configured per Modbus master.
Max. length of the request telegram	With HIMA-specific request telegrams 1100 bytes, see Chapter 7.5.2.
Max. size of process data	A total of 128 kB of data can be transmitted and a total of 128 kB of data can be received. i The status bytes of the master and the status bytes of each slave assigned to it must be subtracted from the max. size of process data (128 kB).
Display format of the Modbus data	The HIMax controllers use the big endian format. Example: 32-bit data (e.g., DWORD, DINT):

32-bit data (hex)	0x12345678			
Memory offset	0	1	2	3
Big endian (HIMax)	12	34	56	78
Middle endian (H51q)	56	78	12	34
Little endian	78	56	34	12

Table 161: Modbus Master Properties

According to the standard, a total of three repeaters may be used such that a maximum of 121 slaves are possible per serial master interface.

7.2 Modbus Example

In this example, the HIMA Modbus master exchanges data with a HIMA Modbus slave through Modbus TCP. Both controllers are connected via the Ethernet interface of the communication modules.

i If the Modbus slave and the Modbus master are located in different subnets, the routing table must contain the corresponding user-defined routes.

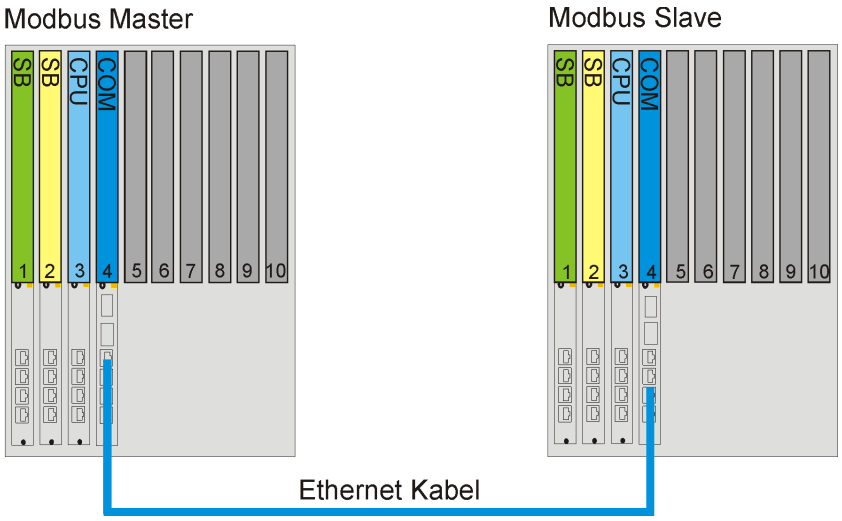


Figure 48: Communication Using Modbus TCP/IP

For this example, the following global variables must be created in SILworX:

Global Variables	Type
Master->Slave_BOOL_00	BOOL
Master->Slave_BOOL_01	BOOL
Master->Slave_BOOL_02	BOOL
Master->Slave_WORD_00	WORD
Master->Slave_WORD_01	WORD
Slave->Master_WORD_00	WORD
Slave->Master_WORD_01	WORD

7.2.1 Configuring the Modbus TCP Slave

To create a new HIMA Modbus Slave

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, select **New, Modbus Slave Set** to add a new Modbus slave set.
3. Select **Edit** on the context menu for the Modbus slave set, open the **Modbus Slave Set Properties**, and retain the default values.
4. Select the **Modbus slave** tab and perform the following actions:
   - Select **COM Module**
   - Activate **Enable TCP**
   - The remaining parameters retain the default values.

To configure the bit input variables of the Modbus slave

i The Boolean variables that the master addresses bit by bit are entered in the Bit Variables tab (function code 1, 2, 5, 15).

1. From the context menu for the Modbus slave, click **Edit** and then select the **Bit Variables** tab.
2. Drag the following global variables from the **Object Panel** onto the **Bit Inputs** area.

Bit address	Bit variable	Type
0	Master->Slave_BOOL_00	BOOL
1	Master->Slave_BOOL_01	BOOL
2	Master->Slave_BOOL_02	BOOL

3. Right-click anywhere in the **Register Inputs** area, and then click **New Offsets** to renumber the variable offsets.

To configure the register input variables of the Modbus slave

i The variables that the master addresses 16 register by register are entered in the Register Variables tab (function code 3, 4, 6, 16 ,23).

1. From the context menu for the Modbus slave, click **Edit** and then select the **Register Variables** tab.
2. Drag the following variables from the **Object Panel** onto the **Register Inputs** area.

Register Address	Register variables	Type
0	Master->Slave_WORD_00	WORD
1	Master->Slave_WORD_01	WORD

3. Right-click anywhere in the **Register Inputs** area, and then click **New Offsets** to renumber the variable offsets.

To configure the register output variables of the Modbus slave

1. From the context menu for the Modbus slave, click **Edit** and then select the **Register Variables** tab.
2. Drag the following variables from the **Object Panel** onto the **Register Outputs** area.

Register Address	Register variables	Type
0	Slave->Master_WORD_00	WORD
1	Slave->Master_WORD_01	WORD

3. Right-click anywhere in the **Register Outputs** area, and then click **New Offsets** to renumber the variable offsets.

To check the Modbus TCP slave configuration

1. Open the context menu for the Modbus TCP master and click **Verification**.
2. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

7.2.2 Configuring the Modbus TCP Master

To create the HIMA Modbus Master

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, click **New, Modbus Master** to add a new Modbus master.
3. From the context menu for the Modbus master, **Properties**, **General**.
4. Click **COM Module**.
   The remaining parameters retain the default values.

To create the connection to the Modbus TCP slave in the Modbus master

1. In the structure tree, open **Resource, Protocols, Modbus Master, Ethernet Slaves.**
2. Right-click **Ethernet Slaves,** then click **New**.
3. Select **TCP/UDP Slaves** from the list and click **OK** to confirm.
4. To configure the TCP/UDP slave in the Modbus master:
   - Click **Edit** to assign the system variables, see Chapter 7.5.10.
   - Click **Properties** to configure the properties, see Chapter 7.5.11.
     Enter the **IP address** of the TCP/UDP slave in the slave's properties.

The remaining parameters retain the default values.

To configure the write request telegram for the bit output variable:

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Write Multiple Coils (15)**.
3. Right-click **Write Multiple Coils (15)**, then click **Properties**.
   - Enter **0** in the **start address of the write area**
4. Right-click **Read Multiple Coils (15)**, then click **Edit**.
5. Drag the following variables from the **Object Panel** onto the **Output Variables** tab..

Offset	Bit Variables	Type
0	Master->Slave_BOOL_00	BOOL
1	Master->Slave_BOOL_01	BOOL
2	Master->Slave_BOOL_02	BOOL

6. Right-click anywhere in the **Output Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

To configure the write request telegram for the register output variable:

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Write Multiple Registers (16)**.
3. Right-click **Write Multiple Register (16),** then click **Properties**.
   - Enter **0** in the **start address of the write area**
4. Right-click **Write Multiple Registers (16)**, then click **Edit**.
5. Drag the following variables from the **Object Panel** onto the **Output Variables** tab..

Offset	Register variables	Type
0	Master->Slave_WORD_00	WORD
1	Master->Slave_WORD_01	WORD

6. Right-click anywhere in the **Output Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

To define the request telegram for reading the input variables in the Modbus master

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Read Holding Registers (03)**.
3. Right-click **Read Holding Registers (03)**, then click **Properties**.
   - Enter **0** in the **start address of the read area**.
4. Right-click **Read Holding Registers (03)**, then click **Edit**.
5. Drag the following variables from the **Object Panel** onto the **Input Variables** tab..

Offset	Register variables	Type
0	Slave->Master_WORD_00	WORD
1	Slave->Master_WORD_01	WORD

6. Right-click anywhere in the **Input Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

To check the Modbus TCP master configuration

1. Open the context menu for the Modbus TCP master and click **Verification**.

To check the Modbus TCP master configuration

1. Open the context menu for the Modbus TCP master and click **Verification**.
2. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

To create the code for the controllers

1. Start the code generator for the master and slave resource.
2. Make sure that the code was generated without error.
3. Load the codes into the master and slave controllers respectively.

7.3 Example of Alternative Register/Bit Addressing

In this example, the configuration defined in Chapter 7.2 is extended by 16 Boolean variables in the Register Area. The 16 Boolean variables are read with the request telegram **Write Multiple Coils (15)**, see also Chapter 7.12.

To configure the input variables in the Modbus slave

1. From the context menu for the Modbus slave, click **Edit** and then select the **Register Variables** tab.
2. Drag the 16 new Boolean variables from the **Object Panel** onto the **Register Inputs** area.

Register Address	Register variables	Type
0	Master->Slave_WORD_00	WORD
1	Master->Slave_WORD_01	WORD
2	Master->Slave_BOOL_03 ..._18	BOOL

Add one again

3. Right-click anywhere in the **Register Inputs** area, and then click **New Offsets** to renumber the variable offsets.

To configure the alternative register/bit addressing in the Modbus slave

1. Right-click the Modbus Slave and select **Edit**, and **Offsets,** then activate **Use Alternative Register/Bit Addressing**.
2. In this example, use the following offsets for the alternative areas:

Register Area Offset Bits Input	1000
Register Area Offset Bits Output	1000
Bit Area Offset Register Input	8000
Bit Area Offset Register Output	8000

i To use the Modbus request telegram **Write Multiple Coils (15)** to access the Boolean variables in the **Register Variables** area, the variables must be mirrored in the **Bit Variables** area.

To configure the write request telegram for the output variable (BOOL) in the Modbus master

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Write Multiple Coils (15)**.
3. Right-click **Write Multiple Coils (15)**, then click **Properties**.
   - Enter **8032** in the **start address of the write area**
4. Right-click **Read Multiple Coils (15)**, then click **Edit**.
5. Drag the following variables from the **Object Panel** onto the **Output Variables** tab..

Offset	Mirrored Register Variable	Type
0 to 15	Master->Slave_BOOL_03..._18	BOOL

6. Right-click anywhere in the **Output Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

7.4 Menu Functions of the HIMA Modbus Master

7.4.1 Edit

The **Edit** dialog box for the Modbus master contains the following tab:

System Variables

The **System Variables** tab contains system variables that are required to control the Modbus Master and evaluate its state from within the user program.

Element	Description
Slave Connection Error Count	Number of faulty connections with Modbus slaves that are in the "Activated" state. Deactivated Modbus slaves are not taken into account.
Modbus Master Activation Control	Stop or start the Modbus master from within the user program. 0:START 1:STOP
Modbus Master Bus Error	Bus error on RS485, e.g., telegram error (code unknown etc), length error.
Modbus Master State	It indicates the current protocol state: 1: OPERATE 0: OFFLINE
Reset All Slave Errors	A change from FALSE to TRUE resets all slave and bus errors.

Table 162: System Variables for the Modbus Master

7.4.2 Properties

The **Properties** function on the context menu for the Modbus master opens the *Properties* dialog box.

The dialog box contains the following tabs:

General

The Tab **General** contains the name and a description for the Modbus master. This tab is also used to set the parameters for specifying whether the Modbus master should also operate as a TCP and/or a UDP gateway.

Element	Description
Type	Modbus master
Name	Name for the Modbus master
Description	A description for the Modbus master.
Module	Selection of the COM module within which the protocol is processed.
Use Max CPU Load	Activated: Use CPU load limit from the Max. CPU Load [%] field. Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%
Enable TCP Gateway	If the TCP Modbus gateway is enabled, at least one Modbus RS485 interface must be configured.
TCP Server Port	Standard: 502 Additional TCP ports may also be configured. Observe the port assignment provided by the ICANN *(Internet Corporation for Assigned Names and Numbers)*.
Maximum Number of TCP connections operating as server.	Maximum number of TCP connections opened simultaneously and operating as server. Range of values:1 ... 64 Default value: 5
Enable UDP gateway	If the UDP Modbus gateway is enabled, at least one Modbus RS485 interface must be configured.
UDP Port	Standard: 502 Additional UDP ports may also be configured. Observe the port assignment provided by the ICANN (Internet Corporation for Assigned Names and Numbers).
Maximum length of the queue	Length of the gateway queue for other masters' request telegrams that have not been answered yet. This option is only taken into account if a gateway has been activated. Range of values: 1...20 Default value: 3

Table 163: General Properties of the Modbus Master

CPU/COM

The default values of the parameters provide the fastest possible data exchange of Modbus data between the COM module (COM) and the processor module (CPU) within the HIMax controller.

These parameters should only be changed if it is necessary to reduce the COM or CPU load for an application, and the process allows this change.

i Only experienced programmers should modify the parameters.

Increasing the COM and CPU refresh rate means that the effective refresh rate of the Modbus data is also increased. The system time requirements must be verified.

Element	Description
Refresh Rate [ms]	Refresh rate in milliseconds at which the COM and CPU exchange protocol data. If the *Refresh Rate* is zero or lower than the cycle time for the controller, data is exchanged as fast as possible. Range of values: 0 ... (2^{31}-1) Default value: 0
Within one cycle	Activated: Transfer of all protocol data from the CPU to the COM within a CPU cycle. Deactivated: Transfer of all protocol data from the CPU to the COM, distributed over multiple CPU cycles, each with 1100 byte per data direction. This can also allow lowering the cycle time of the controller. Default value: Activated

Table 164: Parameters of COM/CPU

7.5 Modbus Function Codes (Request Telegrams)

The Modbus function codes (request telegrams) allow the user to write and read variables in both directions. Individual variables or several consecutive variables can be written or read.

To create a new request telegram for a TCP/UDP slave

1. In the structure tree, open **Resource, Protocols, Modbus Master, Ethernet Slaves**, and then select a **TCP/UDP Slave**.
2. Right-click **TCP/UDP slaves,** then click **New**.
3. In the **New Object** dialog box, click a **Request Telegram**.

To create a new request telegram for a gateway slave

1. In the structure tree, open **Resource, Protocols, Modbus Master, Modbus Gateway**, , and then click a **gateway slave**.
2. Right-click **Gateway Slave,** then click **New**.
3. In the **New Object** dialog box, click a **Request Telegram**.

To create a new request telegram for a RS485 Modbus slave

1. In the structure tree, open **Resource, Protocols, Modbus Master, Serial Modbus**, , and then click a **Modbus slave**.
2. Right-click **Modbus Slave,** then click **New**.
3. In the **New Object** dialog box, click a **Request Telegram**.

7.5.1 Modbus Standard Function Codes

The HIMA Modbus master supports the following Modbus standard function codes:

Element	Code	Type	Description
READ COILS	01	BOOL	Read several variables (BOOL) from the slave.
READ DISCRETE INPUTS	02	BOOL	Read several variables (BOOL) from the slave.
READ HOLDING REGISTERS	03	WORD	Read several variables of any type from the slave.
READ INPUT REGISTERS	04	WORD	Read several variables of any type from the slave.
WRITE SINGLE COIL	05	BOOL	Write one single signal (BOOL) in the slave.
WRITE SINGLE REGISTER	06	WORD	Write one single signal (WORD) in the slave.
WRITE MULTIPLE COILS	15	BOOL	Write several variables (BOOL) in the slave.
WRITE MULTIPLE REGISTERS	16	WORD	Write several variables of any type in the slave.
READ WRITE HOLDING REGISTERS	23	WORD	Write and read several variables of any type in and from the slave.

Table 165: Modbus Function Codes

i (For more information on Modbus, refer to the *Modbus Application Protocol Specification* www.modbus.org)

7.5.2 HIMA-Specific Function Codes

HIMA-specific function codes corresponds to the standard Modbus function codes. The two differences are the maximum permissible process data length of 1100 bytes and the format of the request and response headers.

Element	Code	Type	Description
Read Coils Extended	100 (0x64)	BOOL	Correspond to function code 01. Read several variables (BOOL) from the slave's import or export[1)] area. Maximum length of the process data: 1100 bytes.
Read Discrete Inputs Extended	101 (0x65)	BOOL	Correspond to function code 02. Read several variables (BOOL) from the slave's export area. Maximum length of the process data: 1100 bytes.
Read Holding Registers Extended	102 (0x66)	WORD	Correspond to function code 03. Read several variables of any type from the slave's import or export[1)] area. Maximum length of the process data: 1100 bytes.
Read Input Registers Extended	103 (0x67)	WORD	Correspond to function code 04. Read several variables of any type from the slave's export area. Maximum length of the process data: 1100 bytes.
Write Multiple Coils Extended	104 (0x68)	BOOL	Correspond to function code 15. Write several variables (BOOL) in the slave's import area. Maximum length of the process data: 1100 bytes.
Write Multiple Registers Extended	105 (0x69)	WORD	Correspond to function code 16. Write several variables of any type in the slave's import area. Maximum length of the process data: 1100 bytes.
Read/Write Multiple Registers Extended	106 (0x6A)	WORD	Correspond to function code 23. Write and read several variables of any type in and from the slave's import or export area. Maximum length of the process data: 1100 bytes (request telegram from the master Modbus Master). 1100 bytes (response to the Master).

Format of Request and Response Header

The request and response header of the HIMA-specific Modbus function codes are structured as follows:

Code	Request	Response
100 (0x64)	1 byte function code 0x64 2 bytes start address 2 bytes number of coils 1...8800(0x2260)	1 byte function code 0x64 2 bytes number of bytes = N N bytes coil data (8 coils are packed in one byte)
101 (0x65)	1 byte function code 0x65 2 bytes start address 2 bytes number of discrete inputs 1...8800 (0x2260)	1 byte function code 0x65 2 bytes number of bytes = N N bytes discrete inputs data (8 discrete inputs are packed in one byte)
102 (0x66)	1 byte function code 0x66 2 bytes start address 2 bytes number of registers 1...550 (0x226)	1 byte function code 0x66 2 bytes number of bytes = N N bytes register data
103 (0x67)	1 byte function code 0x67 2 bytes start address 2 bytes number of registers 1...550 (0x226)	1 byte function code 0x67 2 bytes number of bytes = N N bytes register data
104 (0x68)	1 byte function code 0x68 2 bytes start address 2 bytes number of coils 1...8800(0x2260) 2 bytes number of bytes = N N bytes coil data	1 byte function code 0x68 2 bytes start address 2 bytes number of coils 1...8800(0x2260)
105 (0x69)	1 byte function code 0x69 2 bytes start address 2 bytes number of registers 1...550 (0x226) 2 bytes number of bytes = N N bytes register data	1 byte function code 0x69 2 bytes start address 2 bytes number of registers 1...550 (0x226)
106 (0x6A)	1 byte function code 0x6a 2 bytes read start address 2 bytes number of read registers 1...550 (0x226) 2 bytes write start address 2 bytes number of write registers 1...550 (0x226) 2 bytes number of write bytes = N N bytes register data	1 byte function code 0x6a 2 bytes number of bytes = N N bytes register data

7.5.3 Read Request Telegrams

The read function codes are used to read variables from the slave.

In addition to the Modbus function, a Modbus master's request telegram also contains the start address for the read/write area.

To read variables, the Modbus master sends a *Read Request Telegram* to the Modbus slave.

The Modbus slave responds to the Modbus master sending back a response telegram with the variables required.

To configure a read request telegram

1. In the structure tree, click the **Request Telegram** to be configured.
2. Right-click the **request telegram**, then click **Edit**.
3. Select the global variable that should be used as Modbus receive variable and drag it from the **Object Panel** onto anywhere in the **Input Signals** area.
4. Repeat these steps for every further Modbus receive variable.
5. Right-click anywhere in the **Inputs Signals** area, and then click **New Offsets** to renumber the variable offsets.

The following *Read Request Telegrams* are available:

Read Coils (01) and Extended (100)

Read several variables (BOOL) from the slave.

Element	Description
Type	Modbus Function Read Coils
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the read area	0 ...65535

Table 166: Request Telegram Read Coils

Read Discrete Inputs (02) and Extended (101)

Read several variables (BOOL) from the slave.

Element	Description
Type	Modbus Function Read Discrete Inputs
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the read area	0 ...65535

Table 167: Request Telegram Read Discrete Inputs

Read Holding Registers (03) and Extended (102)

Read several variables of any type from the slave.

Element	Description
Type	Modbus Function Read Holding Registers
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the read area	0 ...65535

Table 168: Request Telegram Read Holding Registers

Read Input Registers (04) and Extended (103)

Read several variables of any type from the slave.

Element	Description
Type	Modbus Function Read Input Registers
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the read area	0 ...65535

Table 169: Request Telegram Read Input Registers

Read/Write Request Telegram

For reading and writing variables, the Modbus master sends a *Read/Write Request Telegram* to the Modbus Slave.

First, the Modbus master writes the write variables into the defined import area of the Modbus slave.

Afterwards, the Modbus master reads the read signals from the defined export area of the Modbus slave.

i In the Read/Write Request Telegram, the Write and Read functions are independent of one another.

However, the *Read/Write Request Telegram* is often used such that the variables written by the Modbus master are read back. This ensures that the transferred variables were written correctly.

To configure a read/write request telegram

1. In the structure tree, click the **Request Telegram** to be configured.
2. Right-click the **request telegram**, then click **Edit**.

To configure the read variables

1. In the **Object Panel**, select the global variable that should be connected to one new Modbus receive variable and drag it onto the **Global Variable** column of the Modbus receive variable.
2. Repeat these steps for every further Modbus receive variable.
3. Right-click anywhere in the **Inputs Signals** area, and then click **New Offsets** to renumber the variable offsets.

To configure the write variables

1. In the **Object Panel**, select the global variable that should be connected to one new Modbus send variable and drag it onto the **Global Variable** column of the Modbus send variable.
2. Repeat these steps for every further Modbus send variable.
3. Right-click anywhere in the **Outputs Signals** area, and then click **New Offsets** to renumber the variable offsets..

Read Write Holding Register (23) and Extended (106)

Write and read several variables of any type in and from the slave's import area.

Element	Description
Type	Modbus function *Read Write Holding Registers*
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the read area	0 ...65535
Start address of the write area	0 ...65535

Table 170: Read Write Holding Register

7.5.4 Write Request Telegram

Using the write function codes, variables may only be written in a slave's import area.

In addition to the Modbus function, a Modbus master's request telegram also contains the start address for the read/write area.

To write variables, the Modbus master sends a *Write Request Telegram* to the Modbus slave.

The Modbus slave writes the received variables into its import area.

The variables that a Modbus master writes to a Modbus slave must be entered in the *Variable Connections* dialog box for a *write request telegram*.

To configure a write request telegram

1. In the structure tree, click the **Request Telegram** to be configured.
2. Right-click the **request telegram**, then click **Edit**.
3. Select the global variable that should be used as Modbus send variable and drag it from the **Object Panel** onto anywhere in the **Send Signals** area.
4. Repeat these steps for every further Modbus send variable.
5. Right-click anywhere in the **Send Signals** area, and then click **New Offsets** to renumber the variable offsets.

The following *Write Request telegrams* are available*:*

Write Multiple Coils (15) and Extended (104)

Write several variables (BOOL) in the slave's import area.

Element	Description
Type	Modbus function *Write Multiple Coils*
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the write area	0 ...65535

Table 171: Request Telegram Write Multiple Coils

Write Multiple Registers (16) and Extended (105)

Write several variables of any type in the slave's import area.

Element	Description
Type	Modbus function *Write Multiple Registers*
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the write area	0 ...65535

Table 172: Request Telegram Write Multiple Registers

Write Single Coil (05)

Write one single variable (BOOL) in the slave's import area.

Element	Description
Type	Modbus function *Write Single Coil*
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the write area	0 ...65535

Table 173: Request Telegram Write Single Coil (05)

Write Single Register (06)

Write one single variable (WORD) in the slave's import area.

Element	Description
Type	Modbus function *Write Single Register*
Name	Any unique name for the Modbus function
Description	Description for the Modbus function
Start address of the write area	0 ...65535

Table 174: Request Telegram Write Single Register

7.5.5 Ethernet Slaves (TCP/UDP Slaves)

The Modbus masters can communicate with up to 64 TCP/IP and 247 UDP/IP slaves.

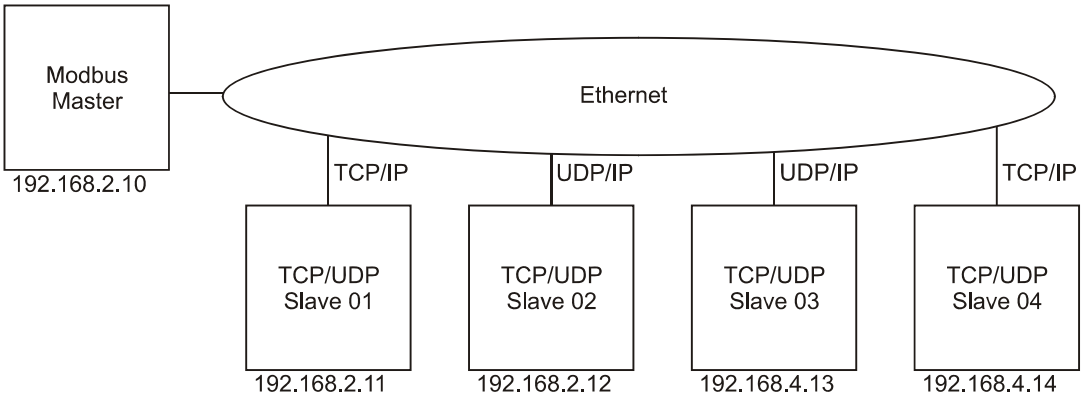


Figure 49: Modbus Network

To create a new connection to a TCP/UDP slave within the Modbus master

1. In the structure tree, open **Resource, Protocols, Modbus Master, Ethernet Slaves.**
2. Right-click **Ethernet Slaves,** then click **New**.
3. Select **TCP/UDP Slaves** from the list and click **OK** to confirm.
4. To configure the TCP/UDP slave in the Modbus master:
   click **Edit** to assign the system variables, see Chapter 7.5.6**.**
   click **Properties** to configure the properties, see Chapter 7.5.7.

i If the TCP/UDP slaves and the Modbus master are located in different subnets, the routing table must contain the corresponding user-defined routes.

7.5.6 System Variables for TCP/UDP Slaves

The *System Variables* tab contains system variables that are required to control the TCP/UDP slave and evaluate its state from within the user program.

The following status variables can be used to evaluate the TCP/UDP slave status from within the user program:

Element	Description
Modbus Slave Activation Control	The user program activates or deactivates the TCP/UDP slave using this function. 0: Activate 1: Deactivate
Modbus Slave Error	Error Code The error codes 0x01...0x0b correspond to the Exception Codes of the Modbus protocol specification. 0x00: No error Exception Codes: 0x01: Invalid function code 0x02: Invalid addressing 0x03: Invalid data 0x04: (not used) 0x05: (not used) 0x06: Device busy (only gateway) 0x08: (not used) 0x0a: (not used) 0x0b: No Response from Slave (only gateway) HIMA-specific codes: 0x10: Defective frame received 0x11: Frame with wrong transaction ID received 0x12: Unexpected response received 0x13: Response about wrong connection received 0x14: Wrong response to a write request 0xff: Slave Timeout
Modbus Slave State	Connection status of the TCP/UDP slave: 0: Disabled 1: Not connected 2: Connected

Table 175: System Variables for TCP/UDP Slaves

7.5.7 TCP/UDP Slave Properties

To configure the connection to the TCP/UDP slave, the following parameters must be set in the Modbus master.

Element	Description
Type	TCP/UDP slave
Name	Any unique name for the TCP/UDP slave
Description	Any unique description for the TCP/UDP slave
IP Address	IP address of the TCP/UDP slave
Port	Standard: 502 Additional TCP/UDP ports may also be configured. Observe the port assignment provided by the ICANN (Internet Corporation for Assigned Names and Numbers).

Type of communication IP protocol	TCP or UDP Default value: TCP
TCP connection only on demand	If the type of the transmission protocol is TCP, the user can set here whether the connection to this slave should be automatically removed after each data exchange: TRUE: Remove connection. FALSE: Do not remove connection Default value: FALSE
Master-Slave Data Exchange [ms]	Time interval for exchanging data with this slave 1 to (2^{31}-1). If the *Maximal Number of Retries* was exceeded and the slave could not be reached, the value for *Master-Slave Data Exchange* is set four times higher.
Maximum Number of Resend	Maximal number of send retries if the slave does not respond. The number of resends can be freely set (0 ...65535). With TCP/IP, the value is always set to 0 and cannot be changed. HIMA recommends setting a value between 0 and 8.
Receive Timeout [ms]	Receive Timeout [ms] for the slave. Once this time period has expired, a resend is attempted.

Table 176: Configuration Parameters

7.5.8 Modbus Gateway (TCP/UDP Gateway)

The Modbus master can operate as Modbus gateway. In this mode, master requests that the gateway receives via Ethernet are forwarded to the RS485 und Ethernet slaves connected to the gateway. Accordingly, the slave's responses are forwarded to the Modbus master through the gateway.

Up to 121 serial Modbus slaves can be addressed per serial interface.

The slave's address ranges from 1 to 247. Even if only the Modbus gateway is used, a Modbus master license is required for Modbus master 2 (Modbus gateway).

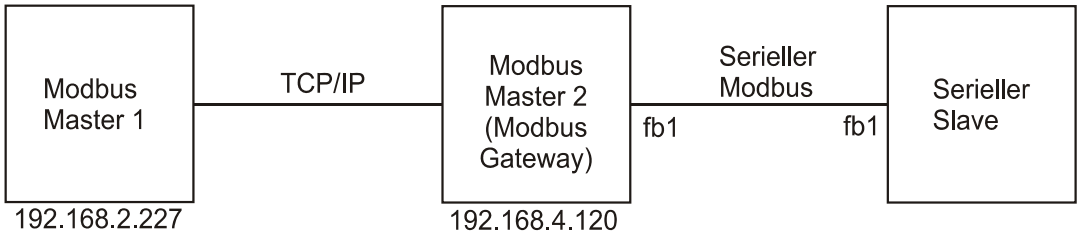


Figure 50: Modbus Gateway

i If the Modbus gateway and the Modbus master are located in different subnets, the routing table must contain the corresponding user-defined routes.

Modbus Master 1

To create the connection to the Modbus slave within Modbus master 1

1. In the structure tree, open **Resource, Protocols, Modbus Master**
2. Right-click **Modbus Master,** then click **New**.
3. Select **Modbus Gateway** from the list and click **OK** to confirm.
4. To configure the Modbus gateway in Modbus Master 1:
   click **Properties** to configure the properties, see Chapter 7.5.15.
   In the properties, enter the **IP Address** for Modbus Master 2 (Modbus gateway).

To create the connection to the gateway slave within Modbus Master 1

In Modbus Master 1, the serial slave must be created as gateway slave.

1. In the structure tree, open **Resource**, **Protocols**, **Modbus Master**, **Modbus Gateway**.
2. Right-click **Modbus Gateway,** then click **New**.
3. Select **Gateway Slave** from the list and click **OK** to confirm.
4. To configure the gateway slave in Modbus Master 1:
   click **Edit** to assign the system variables, see Chapter 7.5.10.
   click **Properties** to configure the properties, see Chapter 7.5.11.
   In the slave's properties, enter the **serial address** for the gateway slave.

To define input and output variables to the serial slave in Modbus Master 1

1. Right-click **Gateway Slave,** then click **New**.
2. Select the required **request telegrams** from the list.
3. Right-click the corresponding **request telegram**, then click **Edit**. Enter the input or output variables in the *Process Variables* tab.

Modbus Master 2 (Modbus Gateway):

The gateway function must be enabled in the properties pf Modbus Master 02. The gateway slaves configured in Master 01 is connected to the serial slaves.

To activate the gateway function in Modbus Master 2

1. In the structure tree, open **Resource, Protocols, Modbus Master**
2. Right-click **Modbus Master,** then click **Properties**.
3. Activate the **Enable TCP Gateway** parameter to allow that the Modbus master can additionally operate as TCP gateway.
4. Activate the **Enable UDP Gateway** parameter to allow that the Modbus master can additionally operate as UDP gateway.

To configure the serial Modbus in Modbus Master 2

1. In the structure tree, open **Resource, Protocols, Modbus Master**
2. Right-click **Modbus Master,** then click **New**.
3. Select **Serial Modbus** from the list and click **OK** to confirm.
4. Select **Properties** to configure the **serial Modbus,** then enter the interface, the baud rate, etc.

To configure the connection to the serial slave in Modbus Master 2

1. In the structure tree, open **Resource**, **Protocols**, **Modbus Master**, **Serial Modbus**.
2. Right-click **Serial Modbus**, then click **New**.
3. Select **Modbus Slave** from the list and click **OK** to confirm.
4. Select **Properties** to configure the **Modbus Slave,** then enter the **Slave Address** for the serial slave.

Serial Slave

To configure the serial Modbus slave

1. In the structure tree, open **Resource, Protocols, Modbus Slave**
2. Right-click **Modbus Slave,** then click **Edit**.
3. Select **Properties** to configure the **Modbus Slave,** then enter the **Slave Address** for the serial slave.

7.5.9 Gateway Properties

The Modbus gateway allows the Modbus master to communicate with its Modbus slave.

To configure the connection to the Modbus gateway, the following parameters must be set in the Modbus master.

Element	Description
Type	Modbus Gateway
Name	Any unique name for the gateway
Description	Any unique description for the TCP/UDP slave
Communication IP Protocol	TCP or UDP Default value: TCP
IP Address	Gateway's IP address that the Modbus Master should use to communicate with its Modbus slave. Default value: (0.0.0.0)
Port	Default value: 502

Table 177: Connection Parameters for the Modbus Gateway

7.5.10 System Variables for the Gateway Slave

The **System Variables** Editor contains the following three status variables (system variables).

Element	Description
Modbus Slave Activation Control	Using this function, the user program can enable or disable the gateway slave. 0: Activate 1: Deactivate
Modbus Slave Error	Parameters: the same as for TCP/UDP slave, see Chapter7.5.6.
Modbus Slave State	Connection status of the gateway slave: 0: Disabled 1: Not connected 2: Connected

Table 178: Status Variables for the Gateway Slave

7.5.11 Gateway Slave Properties

To configure the connection to the gateway slave, the following parameters must be set in the Modbus master.

Element	Description
Type	Gateway Slave
Name	Any unique name for the gateway slave
Description	Any unique description for the gateway slave
Slave Address	1 ...247
The remaining parameters are the same as for TCP/UDP slave, see Chapter7.5.7.	

Table 179: Connection Parameters for the Gateway Slave

7.5.12 Serial Modbus

The Modbus masters can communicate with up to 247 serial slaves. According to the standard, a total of three repeaters may be used such that a maximum of 121 stations are possible per serial interface on a master.

i For more information on the pin assignment of the HIMax COM module's D-sub connectors (fb1, fb2), refer to Chapter 3.7.1.

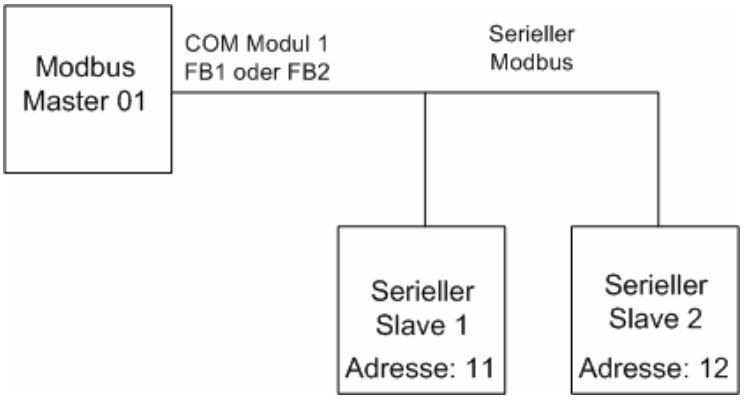


Figure 51: Serial Modbus

The HIMA Modbus master supports data transfer in RTU format (Remote Terminal Unit).

The RTU telegram frame starts and ends with the idle characters set by the user (Default value: 5 idle characters).

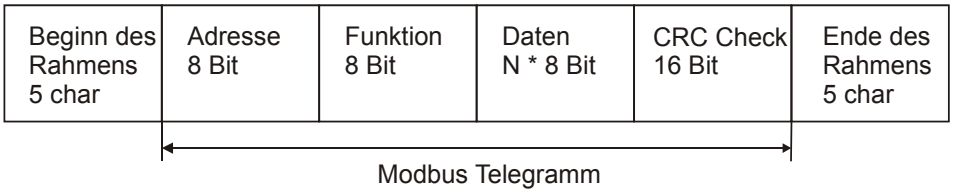


Figure 52: Modbus Telegram

To create a serial Modbus within the Modbus master

1. In the structure tree, open **Resource**, **Protocols**, **Modbus Master**, **Serial Modbus**.
2. Right-click **Serial Modbus,** then click **New**.
3. Select **Modbus Slave** from the list and click **OK** to confirm.
4. To configure the Modbus slave in the Modbus master:
   click **Edit** to assign the system variables, see Chapter 7.5.14.
   click **Properties** to configure the properties, see Chapter 7.5.15.

7.5.13 Serial Modbus Properties

To configure the serial Modbus master, the following parameters must be set.

Element	Description
Type	Serial Modbus
Name	The serial Modbus name may be selected by the user
Description	Any unique description for the serial Modbus
Interface	Fieldbus interface which should be used for the Modbus master (fb1, fb2).
Baud rate [bps]	Transfer rate for RS485 Possible values: 38400 bit/s 19200 bit/s 9600 bit/s 4800 bit/s 2400 bit/s 1200 bit/s 600 bit/s 300 bit/s Default value: 38400
Parity	none Odd Even Default value: Even
Stop Bits	Standard (adapts the number of stop bits to the parity: with parity = 1 stop bit, no parity = 2 stop bits) One stop bit Two stop bits Default value: Default
Number of Idle Chars	Number of idle characters at the start and the end of a RTU telegram frame. Range of values: 0 ... 65535 Default value: 5 characters

Table 180: Parameters for the Serial Modbus Master

7.5.14 System Variables for the Modbus Slave

The Edit Editor contains three status variables (system variables).

Element	Description
Modbus Slave Activation Control	Activate or deactivate the Modbus slave in the user program. 0: Activate 1: Deactivate
Modbus Slave Error	Parameters: the same as for TCP/UDP slave, see Chapter 7.5.7.
Modbus Slave State	Connection status of the Modbus slave: 0: Disabled 1: Not connected 2: Connected

Table 181: System Variables in the Modbus Slave

7.5.15 Modbus Slave Properties

To configure the connection to the serial slave, the following parameters must be set in the Modbus master.

Element	Description
Type	Modbus slave
Name	The Modbus slave name may be selected by the user
Description	Any unique description for the Modbus Slave
Slave Address	1 ...247
The remaining parameters are the same as for TCP/UDP slave, see Chapter7.5.7.	

Table 182: Connection Parameters for the Modbus Master

i In the serial Modbus slave, the Receive Timeout depends on the transfer rate which has been set.

If the baud rate is 19200 [bit/s] or higher, the default value for Receive Timeout may be used. If the baud rate is lower than 19200 [bit/s], the value for Receive Timeout must be increased.

7.6 Control Panel (Modbus Master)

The Control Panel can be used to verify and control the settings for the Modbus master. Details about the master's current state (e.g., master state, etc.) are displayed.

To open the Control Panel for monitoring the Modbus Master

1. In the structure tree, click **Resource**.
2. Click **Online** on the **Action Bar**.
3. In the **System Log-in** window, enter the access data to open the Control Panel for the resource.
4. In the structure tree associated with the Control Panel, select **Modbus Master**.

7.6.1 Context Menu (Modbus Master)

The following commands can be chosen from the context menu for the selected Modbus master:

Offline

This command is used to stop the Modbus master.

Operate

This command is used to start the Modbus master.

Reset statistical data

Reset the statistical data (e.g., number of bus errors, min./max. cycle time etc.) to 0.

7.6.2 View Box (Modbus Master)

The view box displays the following values of the selected Modbus master.

Element	Description
Name	Modbus master name
Master State	It indicates the current protocol state: OPERATE OFFLINE
Bus Error Count	Counter for bus errors
Disturbed Connections	Disturbed connection count
CPU Load (planned)	See Chapter 7.4.2
CPU Load (actual)	

Table 183: View Box of the Modbus Master

7.7 Control Panel (Modbus Master->Slave)

The Control Panel is used to verify and activate/deactivate the settings for the communication partner. Details about the current status of the communication partner (e.g., slave state, etc.) are displayed.

To open Control Panel for monitoring the Modbus connection

1. In the structure tree, click **Resource**.
2. Click **Online** on the **Action Bar**.
3. In the **System Log-in** window, enter the access data to open the Control Panel for the resource.
4. In the structure tree associated with the Control Panel, select **Modbus Master,** and then click a **Slave**.

7.8 FBx LED Function in the Modbus Master

The COM module indicates the state of the local Modbus Master protocol using one of the LEDs assigned to the fieldbus interface. The states of these LEDs are specified in the following table.

FBx LED	Color	Description
OFF	Yellow	The Modbus master protocol is not active! I.e. the controller is in the STOP state or no Modbus master is configured.
Blinking	Yellow	The Modbus master protocol is active and is exchanging data with the Modbus slave.
OFF	Red	The Modbus master protocol is not disturbed.
Blinking	Red	The following events result in a malfunction. ▪ Incorrect response or error message from the slave received ▪ Timeout for one or more slaves ▪ Calculating time budget exceeded If no faults occur for a period longer than 5 seconds, the state changes to "Protocol not disturbed".

Table 184: FBx LED in the MODBUS Master

7.9 HIMA Modbus Slave

The HIMA Modbus slave can simultaneously use the serial interface (RS485) and the TCP/UDP (Ethernet) to operate various Modbus masters.

Equipment and System Requirements

Element	Description
HIMA controller	HIMax with COM module
Processor module	The Ethernet interfaces on the processor module may not be used for Modbus TCP.
COM module	Ethernet 10/100BaseT Pin assignment of the D-sub connectors FB1 and FB2 If Modbus RTU is used, the serial fieldbus interface (FB1 or FB2) used on the COM module must be equipped with an optional HIMA RS485 submodule. For more information on the interface assignment, see Chapter 3.7.
Activation	Each of the two Modbus slave functions must be enabled individually, see Chapter 3.5. Modbus Slave RTU (RS485) Modbus Slave TCP

Table 185: Equipment and System Requirements for the HIMA Modbus Slave

Modbus Slave (Properties)

Element	Description
Modbus slave	One Modbus slave can be configured for each COM module.
Redundancy	A maximum of 10 redundant Modbus slave communication module pairs can be operated in a HIMax system. As long as the Modbus slave communication module pair operates redundantly, the same input and output data is exchanged with the Modbus master via both communication modules.
Number of Master Accesses	RTU: Only one Modbus master can access the slave due to the RS485 transfer mode system. TCP: A maximum of 20 Modbus masters can access the slave.
Max. Size of Send data	128 kB
Max. Size of Receive data	128 kB
Display format of the Modbus data	The HIMax controllers use the big endian format for data. Example: 32-bit data (e.g., DWORD, DINT):

32-bit data (hex)	0x12345678			
Memory offset	0	1	2	3
Big endian (HIMax)	12	34	56	78
Middle endian (H51q)	56	78	12	34
Little endian	78	56	34	12

Table 186: Properties of the Modbus Slave

7.9.1 Configuring the Modbus TCP Slave

To create a new HIMA Modbus Slave

1. In the structure tree, open **Configuration, Resource, Protocols**.

2. On the context menu for protocols, select **New, Modbus Slave Set** to add a new Modbus slave set.
3. Select **Edit** on the context menu for the Modbus slave set, open the **Modbus Slave Set Properties**, and retain the default values.
4. Select the **Modbus slave** tab and perform the following actions:
   - Select **COM Module**
   - Activate **Enable TCP**
   - The remaining parameters retain the default values.

i Chapter 7.2 provides an example of how to configure the connection between an HIMA Modbus TCP slave and an HIMA Modbus TCP master.

7.9.2 Configuring the Redundant Modbus TCP Slave

To create a redundant HIMA Modbus slave

1. In the structure tree, open **Configuration, Resource, Protocols, Modbus Slave Set**.
2. Select **Edit** on the context menu for the Modbus slave set, open the **Modbus Slave Set Properties**, and perform the following actions:
   - Activate **Set Redundant Operation**.

   ☑ The **Modbus Slave Redundant** tab is automatically added.
3. Select the **Modbus Slave Redundant** tab and perform the following actions:
   - Select **COM Module**
   - Activate **Enable TCP**

The remaining parameters retain the default values.

i The send and receive variables assigned in the Modbus slave set are valid for both Modbus slaves.

7.9.3 Rules for Redundant Modbus TCP Slaves

The redundant configuration of the HIMax system is recommended for operating the Modbus slave communication modules redundantly, see the System Manual HI 801 001 E for more details.
Otherwise, the consistent behavior of the Modbus slave communication module pairs towards their external partner (Modbus master) can no longer be ensured once the first error has occurred within the HIMax system.

Slots Allowed for the Redundant Modbus Slave COM Modules

To minimize the risk of potential collisions on the HIMax system bus, the system bus segments (1 to 3) on the base plate must be taken into account. For this reason, the redundant Modbus slave communication modules should only be inserted in the same segment of a base plate in the following slots:

Segment	Slot
1	3...6 (as long as no processor module has been planned)
2	7...14
3	15...18

Table 187: Slots Allowed for the Redundant Modbus Slave COM Modules

Redundant Modbus Slave COM Modules in Different Base Plates

A maximum of two redundant Modbus slave communication modules located in different base plates (0 to 15) may be operated.
Additionally, these redundant Modbus slave communication modules may only be located in adjacent base plates.

NOTE

System malfunction possible!

Only use slots for redundant Modbus slave communication modules in accordance with these rules!

7.10 Menu Functions of the HIMA Modbus Slave Set

The Edit function on the context menu for the Modbus Slave Set opens the *Modbus Slave Set Properties* dialog box. The dialog box contains the following tabs:

7.10.1 Modbus Slave Set Properties

The following parameters for the Modbus slave can be set in the *Modbus Slave Set Properties* tab.

Element	Description
Name	Name of the Modbus slave set
Use Max CPU Load	Activated: Use CPU load limit from the field *Max. CPU Load [%]* Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of the COM module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%
Set Redundant Operation.	Activated: Redundant Operation Deactivated: Mono Operation Default value: Deactivated
Max. Response Time [ms]	Time period after the reception of a request within which the Modbus slave may respond. Range of values: 0...-(2^{31}-1) ms Default value: 0 ms (0 = No limitation)
Area for reading the function codes 1 and 3	This parameter defines from which data field the data should be read for the function codes 1 and 3. Range of values: ▪ Import area ▪ Export area (compatible with 51q)
Area to Read function code 23	The user can specify the Modbus slave's area from which the function code 23 should be read. Import area: The Master has read/write access to the slave's import area. Export area: The Master reads from the slave's export area are writes in the slave's import area. Note: writing and reading take place within a single CPU cycle. This means that the read data was provided during the last CPU cycle.
Initial data at Master timeout	Once the Master timeout has expired, the connection status is set to "not connected". The input variables of the user program are processed depending on the following configuration: Adopt initial data: Input data is reset to its initial values. Retain Last Value: The input data retains its last value.

Behavior on Lost Connection within the CPU	This parameter is used to define wether the value should be adopted from *Initial Data at Master Timeout*. **If a project is converted from versions lower than V3, this value must be deactivated.** Activated: The value is adopted from *Initial Data at Master Timeout*. Deactivated: The input data retains the last value. Default value: Deactivated
Alternative register / bit addressing	Activated Use the alternative addressing Deactivated Do not use the alternative addressing Default value: Deactivated, see Chapter 7.12.
Register Area Offset Bits Input	Range of values: 0 ... 65535 Default value: 0
Register Area Offset Bits Output	Range of values: 0 ... 65535 Default value: 0
Bit Area Offset Register Input	Range of values: 0 ... 65535 Default value: 0
Bit Area Offset Register Output	Range of values: 0 ... 65535 Default value: 0
Refresh Rate [ms]	Refresh rate in milliseconds at which the COM and CPU exchange protocol data. If the *Refresh Rate* is zero or lower than the cycle time for the controller, data is exchanged as fast as possible. Range of values: 0 ... (2^{31}-1) Default value: 0
Within one cycle	Activated: Transfer of all protocol data from the CPU to the COM within a CPU cycle. Deactivated: Transfer of all protocol data from the CPU to the COM, distributed over multiple CPU cycles, each with 1100 byte per data direction. This can also allow lowering the cycle time of the controller. Default value: Activated

Table 188: Modbus Slave Properties Set Tab

7.10.2 Register Variable

(Access Tab)

The variables that the master addresses 16 bit by bit are entered in the Register Variables tab (function code 3, 4, 6, 16, 23, 102, 103, 105, 106).

7.10.3 Bit Variables

(Bit and/or coil access)

The variables that the master addresses bit by bit are entered in the Bit Variables tab (function code 1, 2, 5, 15, 100, 101, 104).

7.10.4 Assigning Send/Receive Variables

All the variables that the Modbus slave receives from the Modbus master are entered in the **Inputs** tab.

To configure the send variables of the Modbus slave

1. In the structure tree, select the **Modbus Slave** that should be configured.
2. Right-click **Modbus Slave**, and then click **Edit**.
3. Select the **Register Variables** or **Bit Variables** tab.
4. Drag one **variable** from the *Object Panel* onto the *Register Outputs* area.
5. Repeat these steps for every further variable that should be defined as send variable for the Modbus Slave.
6. Right-click the *Register Outputs* area, and then click **New Offsets**.

To configure the receive variables of the Modbus slave

1. In the structure tree, select the **Modbus Slave** that should be configured.
2. Right-click **Modbus Slave**, and then click **Edit**.
3. Select the **Register Variables** or **Bit Variables** tab.
4. Drag one **variable** from the *Object Panel* onto the *Register Inputs* area.
5. Repeat these steps for every further variable that should be defined as receive variable for the Modbus Slave.
6. Right-click the *Register Inputs* area, and then click **New Offsets**.

7.10.5 Modbus Slave Set System Variables

The **Modbus Slave Set System Variables** includes the following system variables.

Element	Description
Redundant State	This parameter describes the redundancy state of the redundant Modbus slave communication module pair. 0: Redundant Modbus Slave COM Modules active 1: First Modbus Slave COM Module not active 2: Redundant Modbus Slave COM Module not active 3: Both Modbus Slave COM Modules not active

Table 189: View Box of the Modbus Master

7.10.6 Modbus Slave and Modbus Slave Redundant

The **Modbus Slave** tab contains the two tabs Properties and System Variables.

i The pin assignment of the D-sub connectors (fb1, fb2) is described in the manuals of the corresponding HIMax module.

Properties

Element	Description
Module	Selection of the COM module within which the protocol is processed.
Master Monitoring Time [ms]	Timeout within which the slave must receive at least one request from the master. If the slave receives no request within the timeout, the "Master Connection Status" is set to "not connected". Range of values: 1...2^{31}-1 [ms] Default value : 0=Off
Parameters for the Ethernet interface	
Enable the TCP/IP connection	Activated The TCP/IP connection is enabled Deactivated TCP/IP connection disabled Default value: Deactivated
TCP Port	Default value: 502
Maximum number of TCP Connections	Maximum number of TCP connections opened simultaneously and operating as server. Range of values: 1...20 Default value: 3
UDP Enable	Activated UDP/IP connection is enabled Deactivated UDP/IP connection disabled Default value: Deactivated
UDP Port	Default value: 502
Parameters for the serial interface	
Name	Name of the serial interface
Interface	Selection of the fieldbus interfaces that are available and can be used for the Modbus slave (none, fb1, fb2).
Slave Address	Slave bus address Range of values: 1 ... 247
Baud rate [bps]	Possible value for transfer rate for RS485: 38400 bit/s 19200 bit/s 9600 bit/s 4800 bit/s 2400 bit/s 1200 bit/s 600 bit/s 300 bit/s
Parity	Range of values: ▪ none ▪ Odd ▪ Even Default value: Even
Stop Bits	Range of values: Standard (adapts the number of stop bits to the parity: with parity = 1 stop bit, no parity = 2 stop bits) One stop bit Two stop bits Default value: Default
Number of Idle Chars	Number of idle characters at the start and the end of a RTU telegram frame. Range of value: 1 ... 65535 Default value: 5 characters

Table 190: TCP and UDP Ports Tab for HIMA Modbus Slave

The **System Variables** tab contains system variables that are required to control the Modbus Slave and evaluate its state from within the user program.

Element	Description
Average Concurrent Master Requests	Average number of concurrent master requests
Valid Master Requests	Number of valid master requests since the last reset of all counters or last HIMax controller's start-up.
Master Requests	Total number of master requests since the last reset of all counters or last HIMax controller's start-up.
Master Monitoring Time [ms]	Timeout within which the slave must receive at least one request from the master. If the slave does not receive any request within the timeout period, the *Master Connection Status* is set to **1** *not connected*. Depending on the configuration, the user program's input data is reset to its initial values or it retains its last value.
Master Connection State	FALSE: Not Connected TRUE: Connected
Maximum Concurrent Master Requests	Maximum number of concurrent master requests
Response Timeouts	Number of response timeouts since the last reset of all counters or last HIMax controller's start-up. The response timeout is the maximum time within which the sending station must receive the message acknowledgment.
Reset All Counters	This system variable is used to reset all counters in the user program. A change from 0 to 1 triggers the reset function. Values greater than 1 are treated as 1.
Invalid Master Requests	Number of invalid master requests since the last reset of all counters or last HIMax controller's start-up. An invalid request is a request upon which the Modbus slave sends an error code to the Modbus master. Incorrect transmissions that were already detected and filtered out at the driver level (framing errors, CRC errors, length errors) are not included here, but they are reported through the diagnosis.
Rejected Master Requests	Number of rejected master requests since the last reset of all counters or last HIMax controller's start-up.

Table 191: System Variables Tab for the HIMA Modbus Slave

7.10.7 Modbus Function Codes

The HIMA Modbus slave supports the following Modbus function codes:

Element	Code	Type	Description
READ COILS	01	BOOL	Read several variables (BOOL) from the slave's import or export[1)] area. Maximum length of the process data: 251 bytes.
READ DISCRETE INPUT	02	BOOL	Read several variables (BOOL) from the slave's export area. Maximum length of the process data: 251 bytes.
READ HOLDING REGISTER	03	WORD	Read several variables of any type from the slave's import or export[1)] area. Maximum length of the process data: 250 bytes.
READ INPUT REGISTER	04	WORD	Read several variables of any type from the slave's export area. Maximum length of the process data: 250 bytes.
WRITE SINGLE COIL	05	BOOL	Write one single signal (BOOL) in the slave's import area. Maximum length of the process data: 1 byte
WRITE SINGLE REGISTER	06	WORD	Write one single signal (WORD) in the slave's import area. Maximum length of the process data: 2 bytes.
Diagnostics	08	x	Only sub-code 0: Loop-back function of the slave.
WRITE MULTIPLE COILS	15	BOOL	Write several variables (BOOL) in the slave's import area. Maximum length of the process data: 247 bytes.
WRITE MULTIPLE REGISTER	16	WORD	Write several variables of any type in the slave's import area. Maximum length of the process data: 246 bytes.
READ WRITE MULTIPLE REGISTER	23	WORD	Write and read several variables of any type in and from the slave's import or export area. Maximum length of the process data: 242 bytes (request telegram from the master Modbus Master). 250 bytes (response to the Master).
Read Device Identification	43	x	Transmit the slave identification data to the master.

[1)]The export area can only be selected in HIMA slaves

Table 192: Modbus Function Codes of the HIMA Modbus Slave

In addition to the WORD data type (2 bytes), the function codes 03, 04, 16 and 23 support all other data types.

The start address of the first variable to be transferred and the number of registers/bits of the variables to be transferred must be entered for each request.

Error Codes:

- If the master sends a telegram with unknown function codes, the controller responds with error code 1 (invalid code).
- If the telegram does not match the Modbus slave configuration (i.e., the request telegram does not ends "even" at one variable limit), the slave responds with error code 2 (invalid data).
- If the master sends a telegram with incorrect values (e.g., length fields), the slave responds with error code 3 (invalid value).

Communication only takes place if the COM module is in the RUN state. If the COM module is in any other operating state, the master does not respond to any requests.

Note for Modbus Function: Read Device Identification (43)

The HIMax Modbus slave provides the identification data to the master and supports the following object IDs:

Basic:
0x00 VendorName “HIMA Paul Hildebrandt GmbH + Co KG”
0x01 ProductCode “<Module serial number>”
0x02 MajorMinorRevision “<COM Vx.y CRC>”

Regular:
0x03 VendorUrl “http://www.hima.de”
0x04 ProductName “HIMax”
0x05 ModelName “HIMax”
0x06 UserApplicationName “--------[S.R.S]”

Extended:
0x80 blank “--------”
0x81 blank “--------”
0x82 blank “--------”
0x83 blank “--------”
0x84 blank “--------”
0x85 blank “--------“
0x86 CRC of the file modbus.config “<0x234adcef>“
(configuration file for the Modbus slave protocol in the CPU file system. To compare with the information specified in SILworX, Online/Version comparison).

The following ReadDevice ID Codes are supported:
(1) Read Basic device identification (stream access)
(2) Read regular device identification (stream access)
(3) Read extended device identification (stream access)
(4) Read one specific identification object (individual access)

(For more information on Modbus, refer to the Modbus "Application Protocol Specification" www.modbus.org)

7.10.8 HIMA-Specific Function Codes

HIMA-specific function codes corresponds to the standard Modbus function codes. The only differences are the maximum permissible process data length of 1100 bytes and the format of the request and response headers.

Element	Code	Type	Description
Read Coils Extended	100 (0x64)	BOOL	Correspond to function code 01. Read several variables (BOOL) from the slave's import or export[1)] area. Maximum length of the process data: 1100 bytes.
Read Discrete Inputs Extended	101 (0x65)	BOOL	Correspond to function code 02. Read several variables (BOOL) from the slave's export area. Maximum length of the process data: 1100 bytes.
Read Holding Registers Extended	102 (0x66)	WORD	Correspond to function code 03. Read several variables of any type from the slave's import or export[1)] area. Maximum length of the process data: 1100 bytes.
Read Input Registers Extended	103 (0x67)	WORD	Correspond to function code 04. Read several variables of any type from the slave's export area. Maximum length of the process data: 1100 bytes.
Write Multiple Coils Extended	104 (0x68)	BOOL	Correspond to function code 15. Write several variables (BOOL) in the slave's import area. Maximum length of the process data: 1100 bytes.
Write Multiple Registers Extended	105 (0x69)	WORD	Correspond to function code 16. Write several variables of any type in the slave's import area. Maximum length of the process data: 1100 bytes.
Read/Write Multiple Registers Extended	106 (0x6A)	WORD	Correspond to function code 23. Write and read several variables of any type in and from the slave's import or export area. Maximum length of the process data: 1100 bytes (request telegram from the master Modbus Master). 1100 bytes (response to the Master).

Format of Request and Response Header

The request and response header of the HIMA-specific Modbus function codes are structured as follows:

Code	Request	Response
100 (0x64)	1 byte function code 0x64 2 bytes start address 2 bytes number of coils 1...8800(0x2260)	1 byte function code 0x64 2 bytes number of bytes = N N bytes coil data (8 coils are packed in one byte)
101 (0x65)	1 byte function code 0x65 2 bytes start address 2 bytes number of coils 1 ... 8800 (0x226)	1 byte function code 0x65 2 bytes number of bytes = N N bytes coil data (8 coils are packed in one byte)
102 (0x66)	1 byte function code 0x66 2 bytes start address 2 bytes number of registers 1...550 (0x226)	1 byte function code 0x66 2 bytes number of bytes = N N bytes register data
103 (0x67)	1 byte function code 0x67 2 bytes start address 2 bytes number of registers 1...550 (0x226)	1 byte function code 0x67 2 bytes number of bytes = N N bytes register data
104 (0x68)	1 byte function code 0x68 2 bytes start address 2 bytes number of coils 1...8800(0x2260) 2 bytes number of bytes = N N bytes coil data	1 byte function code 0x66 2 bytes start address 2 bytes number of coils 1...8800(0x2260)
105 (0x69)	1 byte function code 0x69 2 bytes start address 2 bytes number of registers 1...550 (0x226) 2 bytes number of bytes = N N bytes register data	1 byte function code 0x69 2 bytes start address 2 bytes number of registers 1...550 (0x226)
106 (0x6A)	1 byte function code 0x6a 2 bytes read start address 2 bytes number of read registers 1...550 (0x226) 2 bytes write start address 2 bytes number of write registers 1...550 (0x226) 2 bytes number of write bytes = N N bytes register data	1 byte function code 0x6a 2 bytes number of bytes = N N bytes register data

7.11 Addressing Modbus using Bit and Register

The addressing mode adheres to the Modbus addressing standard and only knows the two data lengths bit (1 bit) and register (16 bits) that are used to transfer all the data types allowed.

There are two areas within the Modbus slave: a **Register Area** (inputs and outputs) and a **Bit Area** (inputs and outputs). Both areas are separated from one another and can accept all permitted data types. The difference between these areas resides in the Modbus function codes permitted for accessing these areas.

i The Modbus addressing using bit and register does not guarantee the variable integrity, meaning that any arbitrary portion of a variable can be read or written with this access mode. Variables of type BOOL are stored in a packed format, i.e., each variable of type BOOL is stored as a bit within a byte.

7.11.1 Register Area

The variables in the Register Area are created in the **Register Variables** tab. For more information on how to assign send/receive variables, refer to Chapter 7.10.4.

i To access variables in the Register Area with the Modbus function codes 1, 2, 5, 15, the variables must be mirrored in the Bit Area, see Chapter 7.12.1.

The variables in the Register Area can only be accessed using the Modbus function codes 3, 4, 6, 16, 23. To do this, enter the start address of the first variable in the properties of the function code.

Example: Accessing the Variables in the Register Area of the Modbus Slave

Register variables	Register.Bit	Bit
00_Register_Area_WORD	0.0	0
01_Register_Area_SINT	1.8	16
02_Register_Area_SINT	1.0	24
03_Register_Area_REAL	2.0	32
04_Register_Area_BOOL	4.8	64
05_Register_Area_BOOL	4.9	65
06_Register_Area_BOOL	4.10	66
07_Register_Area_BOOL	4.11	67
08_Register_Area_BOOL	4.12	68
09_Register_Area_BOOL	4.13	69
10_Register_Area_BOOL	4.14	70
11_Register_Area_BOOL	4.15	71
12_Register_Area_BOOL	4.0	72
13_Register_Area_BOOL	4.1	73
14_Register_Area_BOOL	4.2	74
15_Register_Area_BOOL	4.3	75
16_Register_Area_BOOL	4.4	76
17_Register_Area_BOOL	4.5	77
18_Register_Area_BOOL	4.6	78
19_Register_Area_BOOL	4.7	79

Table 193: Register Variables in the Register Area of the Modbus Slave

HIMA Modbus Master Configuration of the Request Telegram

To read the variables 01_Register_Area_SINT to 03_Register_Area_REAL in the Modbus master

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Read Holding Registers (3)**.
3. Right-click **Read Holding Registers (3)**, then click **Properties**.
   - Enter **1** in the **start address of the read area**.
4. Right-click **Read Holding Registers (3)**, then click **Edit**.
5. Drag the following variables from the **Object Panel** onto the **Input Variables** tab..

Register variables	Offset
01_Register_Area_SINT	0
02_Register_Area_SINT	1
03_Register_Area_REAL	2

6. Right-click anywhere in the **Output Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

7.11.2 Bit Area

The variables in the Bit Area are created in the **Bit Variables** tab. For more information on how to assign send/receive variables, refer to Chapter 7.10.4.

i To access variables in the Register Area with the Modbus function codes 3, 4, 6, 16 and 23, the variables must be mirrored in the Bit Area, see Chapter 7.12.2.

The variables in the Bit Area can only be accessed using the Modbus function codes 1, 2, 5, 15. To do this, enter the start address of the first variable in the properties of the function code.

Example: Accessing the Variables in the Bit Area of the Modbus Slave

Bit Variables	Bit	Register.Bit
00_BIT_Area_WORD	0	0.0
01_BIT_Area_SINT	16	1.8
02_BIT_Area_SINT	24	1.0
03_BIT_Area_REAL	32	2.0
04_BIT_Area_BOOL	64	4.8
05_BIT_Area_BOOL	65	4.9
06_BIT_Area_BOOL	66	4.10
07_BIT_Area_BOOL	67	4.11
08_BIT_Area_BOOL	68	4.12
09_BIT_Area_BOOL	69	4.13
10_BIT_Area_BOOL	70	4.14
11_BIT_Area_BOOL	71	4.15
12_BIT_Area_BOOL	72	4.0
13_BIT_Area_BOOL	73	4.1
14_BIT_Area_BOOL	74	4.2
15_BIT_Area_BOOL	75	4.3
16_BIT_Area_BOOL	76	4.4
17_BIT_Area_BOOL	77	4.5
18_BIT_Area_BOOL	78	4.6
19_BIT_Area_BOOL	79	4.7

Table 194: Bit Variables in the Bit Area of the Modbus Slave

HIMA Modbus Master Configuration of the Request Telegram

To read the variables 04_BIT_Area_BOOL to 06_Area_BOOL in the Modbus master

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Read Coils (1)**.
3. Right-click **Read Coils (1)**, then click **Properties**.
   - Enter **64** in the **start address of the read area**.
4. Right-click the request telegram **Read Coils (1)**, then click **Properties**.
5. Drag the following variables from the **Object Panel** onto the **Input Variables** tab..

Bit Variables	Offset
04_BIT_Area_BOOL	0
05_BIT_Area_BOOL	1
06_BIT_Area_BOOL	2

6. Right-click anywhere in the **Output Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

7.12 Offsets for Alternative Modbus Addressing

To access the variables in the **Bit Area** using the Modbus function codes (of type 'register'), the variables must be mirrored in the **Register Area,** and to access the variables in the **Register Area** using the Modbus function codes (of type 'bit'), the variables must be mirrored in the **Bit Area**. The offsets for the mirrored variables are entered in the **Properties/Offsets** tab.

To mirror the variables in the Bit Area and Register Area

1. Right-click the Modbus Slave and select **Edit**, and **Offsets,** then activate **Use Alternative Register/Bit Addressing**.
   ☑ This action mirrors the variables in the corresponding area.
2. Enter the offset for the mirrored variables in the Bit Area and Register Area.

i The existing variables and the corresponding variables mirrored in the B/Register Area must not overlap with respect to the Modbus addresses.

Element	Description / Range of values
Alternative register / bit addressing	Activated Use the alternative addressing Deactivated Do not use the alternative addressing Default value: Deactivated
Register area offset / bit inputs	0 ...65535
Register area offset / bit outputs	0 ...65535
Bit area offset / register inputs	0 ...65535
Bit area offset / register outputs	0 ...65535

Table 195: Offsets Tab for HIMA Modbus Slave

7.12.1 Access to the Register Variables in the Bit Area of the Modbus Slave

To access the Register Area with the Modbus function codes (of type 'bit') 1, 2, 5, 15, the register variables must be mirrored in the **Bit Area**. The offsets for the mirrored register variables must be entered in the **Properties/Offsets** tab.

Example:

Bit area offset / register inputs 8000
Bit area offset / register outputs 8000
The variables mirrored from the Register Area to the Bit Area are located here starting with Bit Address 8000.

Mirrored Register Variables	Bit
00_Register_Area_WORD	8000
01_Register_Area_SINT	8016
02_Register_Area_SINT	8024
03_Register_Area_REAL	8032
04_Register_Area_BOOL	8064
05_Register_Area_BOOL	8065
06_Register_Area_BOOL	8066
07_Register_Area_BOOL	8067
08_Register_Area_BOOL	8068
09_Register_Area_BOOL	8069
10_Register_Area_BOOL	8070
11_Register_Area_BOOL	8071
12_Register_Area_BOOL	8072
13_Register_Area_BOOL	8073
14_Register_Area_BOOL	8074
15_Register_Area_BOOL	8075
16_Register_Area_BOOL	8076
17_Register_Area_BOOL	8077
18_Register_Area_BOOL	8078
19_Register_Area_BOOL	8079

Table 196: Variables Mirrored from the Register Area to the Bit Area

HIMA Modbus Master Configuration of the Request Telegram

To read the variables 04_Register_Area_BOOL to 06_Register_Area_BOOL in the Modbus master

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Read Coils (1)**.
3. Right-click **Read Coils (1)**, then click **Properties**.
   - Enter **8064** in the **start address of the read area**.
4. Right-click the request telegram **Read Coils (1)**, then click **Properties**.
5. Drag the following variables from the **Object Panel** onto the **Input Variables** tab..

Mirrored Register Variables	Offset
04_Register_Area_BOOL	0
05_Register_Area_BOOL	1
06_Register_Area_BOOL	2

6. Right-click anywhere in the **Output Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

7.12.2 Access to the Bit Variables in the Register Area of the Modbus Slave

To access the bit variables with the Modbus function codes (of type 'register') 3, 4, 6, 16, 23, the bit variables must be mirrored in the **Register Area**. The offsets for the mirrored bit variables must be entered in the **Properties/Offsets** tab.

Example:

Register area offset / bit inputs: 1000
Register area offset / bit outputs: 1000
The variables mirrored from the Bit Area to the Register Area are located here starting with Register Address 1000.

Mirrored Bit Variables	Register.Bit
00_BIT_Area_WORD	1000.0
01_BIT_Area_SINT	1001.8
02_BIT_Area_SINT	1001.0
03_BIT_Area_REAL	1002.0
04_BIT_Area_BOOL	1004.8
05_BIT_Area_BOOL	1004.9
06_BIT_Area_BOOL	1004.10
07_BIT_Area_BOOL	1004.11
08_BIT_Area_BOOL	1004.12
09_BIT_Area_BOOL	1004.13
10_BIT_Area_BOOL	1004.14
11_BIT_Area_BOOL	1004.15
12_BIT_Area_BOOL	1004.0
13_BIT_Area_BOOL	1004.1
14_BIT_Area_BOOL	1004.2
15_BIT_Area_BOOL	1004.3
16_BIT_Area_BOOL	1004.4
17_BIT_Area_BOOL	1004.5
18_BIT_Area_BOOL	1004.6
19_BIT_Area_BOOL	1004.7

Table 197: Variables Mirrored from the Bit Area to the Register Area

HIMA Modbus Master Configuration of the Request Telegram

To read the variables 01_BIT_Area_SINT to 03_BIT_Area_REAL in the Modbus master

1. Right-click **TCP/UDP slaves,** then click **New**.
2. From the list, select **Read Holding Registers (3)**.
3. Right-click **Read Holding Registers (3)**, then click **Properties**.
   - Enter **1001** in the **start address of the read area**.
4. Right-click **Read Holding Registers (3)**, then click **Edit**.
5. Drag the following variables from the **Object Panel** onto the **Input Variables** tab..

Mirrored Bit Variables	Offset
01_BIT_Area_SINT	0
02_BIT_Area_SINT	1
03_BIT_Area_REAL	2

6. Right-click anywhere in the **Output Variables** area to open the context menu and click **New Offsets** to renumber the variable offsets.

7.13 Control Panel (Modbus Slave)

The Control Panel can be used to verify and control the settings for the Modbus slave. Details about the slave's current state (e.g., master state, etc.) are displayed.

To open the Control Panel for monitoring the Modbus Slave

1. In the structure tree, click **Resource**.
2. Click **Online** on the **Action Bar**.
3. In the **System Log-in** window, enter the access data to open the Control Panel for the resource.
4. In the structure tree associated with the Control Panel, select **Modbus Slave**.

7.13.1 Context Menu (Modbus Slave)

The following command is available on the context menu for the selected Modbus slave:

Reset statistical data

Reset the statistical data (e.g., min./max. cycle time etc.) to 0.

7.13.2 View Box (Modbus Slave)

The view box displays the following values of the selected Modbus slave.

Element	Description
Name	Modbus slave name
CPU Load (planned) [%]	See Chapter 7.10
CPU Load (actual) [%]	

Table 198: View Box of the Modbus Slave

7.13.3 View Box (Master Data)

The Master Data view box displays the following values.

Element	Description
Name	Name of master data
Interface	Modbus capable interface on the COM module (RS485, Ethernet UDP/TCP)
Requests	Total number of master requests since the last counter reset.
Valid Requests	Number of valid master requests since the last counter reset.
Invalid Requests	Number of invalid master requests since the last counter reset.
Timeout [ms]	Timeout within which the slave must receive at least one request from the master. If the slave receives no request within the timeout, the *Master Connection Status* is set to *not connected*.
Connection State	0 = not monitored Master Request Timeout is zero 1 = Not connected 2 = Connected

Table 199: Master Data View Box

7.14 FBx LED Function in the Modbus Slave

The COM module indicates the state of the local Modbus slave protocol using one of the LEDs assigned to the fieldbus interface. The states of these LEDs are specified in the following table.

FBx LED	Color	Description
OFF	Yellow	The Modbus slave protocol is not active! This means that the controller is in STOP or no Modbus master is configured.
Blinking	Yellow	The Modbus Slave protocol is active and is exchanging data with the Modbus master.
OFF	Red	PROFIBUS DP Slave protocols is not disturbed.
Blinking	Red	The Modbus Slave protocol is disturbed. The following events result in a malfunction. ▪ Unknown function code received ▪ Request with incorrect addressing received ▪ Calculating time budget exceeded If no faults occur for a period longer than 5 seconds, the state changes to *Protocol not disturbed*.

Table 200: FBx LED in the MODBUS Slave

7.14.1 Error Codes of the Modbus TCP/IP Connection

The error codes of the Modbus TCP/IP connection are output in the **Diagnosis** dialog box.

Error Code	Description
35	Operation is blocked
48	Port already in use
50	Network is down
53	Software caused connection abort
54	Peer caused connection abort
55	No buffer space available
60	Operation timed out, connection terminated
61	Connection refused (from peer)
65	No route to peer host

Table 201: Error Codes of Modbus TCP/IP

8 Send & Receive TCP

S&R TCP is a manufacturer-independent, standard protocol for cyclic and acyclic data exchange and does not use any specific protocols other than TCP/IP.

With the S&R TCP protocol, the HIMax controller supports almost every third-party system as well as PCs with implemented socket interface to TCP/IP (for example Winsock.dll).

S&R TCP is compatible with the Siemens SEND/RECEIVE interface and ensures communication with Siemens controllers via TCP/IP. Data is exchanged using the S7 function blocks AG_SEND (FC5) and AG_RECV (FC6).

8.1 System Requirements

Equipment and system requirements

Element	Description
Controller	HIMax with COM module
Processor module	The Ethernet interfaces on the processor module may not be used for S&R TCP.
COM module	Ethernet 10/100BaseT One S&R TCP protocol can be configured for each COM module.
Activation	Software activation code required, see Chapter 3.5.

Table 202: Equipment and System Requirements for the S&R TCP

Properties of the S&R TCP protocol

Element	Description
Safety-related	No
Data Exchange	Cyclic and acyclic data exchange via TCP/IP.
Function Blocks	The S&R TCP function blocks must be used for acyclically exchanging data.
TCP Connections	Up to 32 TCP connections can be configured in one controller, provided that the maximum size of transmission or received data is not exceeded.
Max. size of process data	A total of 128 kB of data can be transmitted and a total of 128 kB of data can be received. i To determine the maximum amount of reference data, the value for all status variables of the configured TCP connections and TCP/SR function blocks must be subtracted from the value for the maximum amount of send data (128 bytes). The data can be freely allocated among several TCP connections.

Table 203: S&R TCP Properties

8.1.1 Creating a S&R TCP Protocol

To create a new S&R TCP protocol

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, click **New, Send/Receive over TCP** to add a new S&R TCP protocol.
3. On the context menu for Send/Receive over TCP, click **Properties.** In the **General** tab, select **COM Module**.

8.2 Example: S&R TCP Configuration

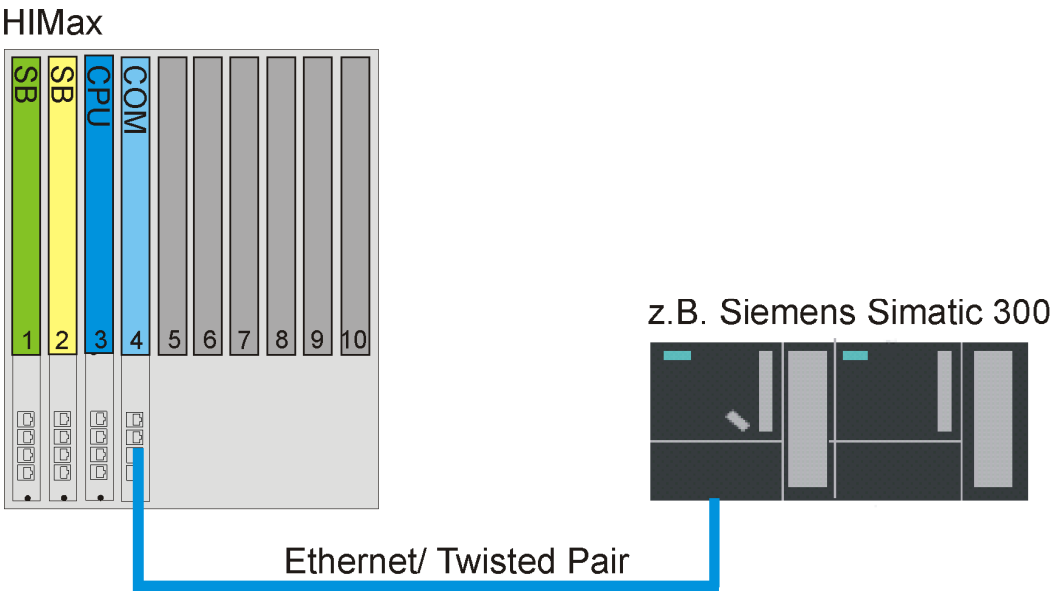


Figure 53: Connecting a HIMax and a Siemens Controller

In this example, the protocol Send/Receive over TCP is installed in a HIMax controller. The HIMax is supposed to cyclically communicate via S&R TCP with a Siemens controller (e.g., SIMATIC 300).

In this example, HIMax (Client) is the active station that establishes the TCP connection to the passive Siemens SIMATIC 300 (Server). Once the connection has been established, both stations are equal and can send and receive data at any point in time.

When connecting the HIMax to the Siemens SIMATIC 300, the following points must be taken into account:

The requirements described in chapter 8.1 System requirements apply for the HIMax.

HIMax and Siemens SIMATIC 300 are connected to one another via Ethernet interfaces.

HIMax and Siemens SIMATIC 300 must be located in the same subnet or must have the corresponding routing settings if a router is used.

In this example, the HIMA controller is supposed to send two BYTES and one WORD to the Siemens SIMATIC 300. The variables are received in the Siemens SIMATIC 300 by the function block AG_RECV (FC 6) and are internally transmitted to the function block AG_SEND (FC 5). Siemens SIMATIC 300 sends the variables (unchanged) back to the HIMax controller using the function block AG_SEND (FC 5).

Once the configuration is completed, the user can verify the variable transmission using the HIMA Force Editor.

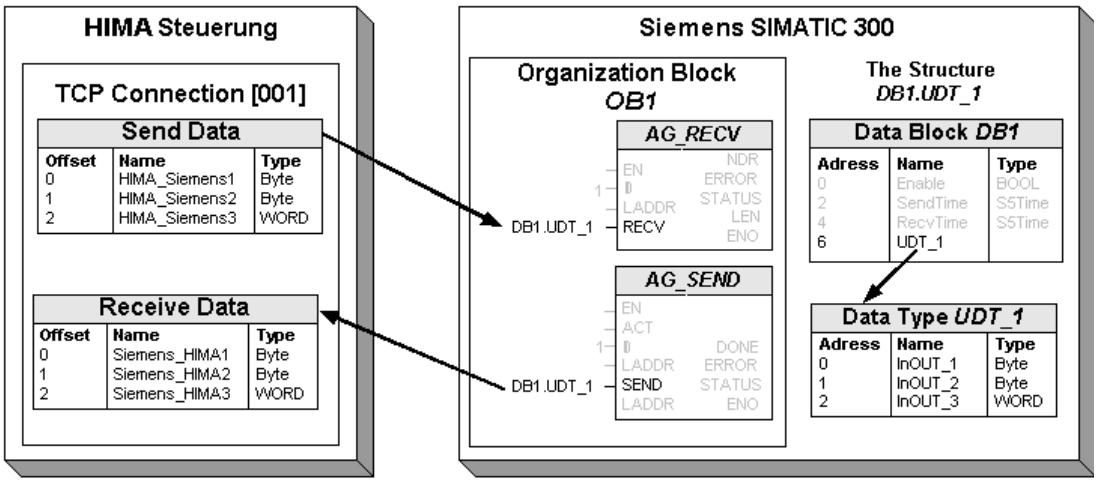


Figure 54: Data Transfer between a HIMax and a Siemens Controller

Description of the HIMax controller configuration

Element	Description
TCP connection [001]	This dialog box contains all parameters required for communicating with the communication partner (Siemens SIMATIC 300).
Send data	The variable offsets and types in the controller must be identical with the variable addresses and types with data type *UDT_1* in the SIMATIC 300.
Receive data	The variable offsets and types in the HIMax controller must be identical with the variable addresses and types with data type *UDT_1* in the SIMATIC 300.

Table 204: HIMax Controller Configuration

Description of the Siemens SIMATIC 300 Configuration

Element	Description
Organization block *OB1*	The function blocks *AG_RECV* (FC6) and *AG_SEND* (FC 5) must be created and configured in the *OB1* organization block.
AG_RECV (FC 6)	The function block *AG_RECV (FC 6)* accepts the data received from the communication partner with data type *DB1.UDT_1.* The inputs *ID* and *LADDR* must be appropriately configured for communication with the communication partner.
AG_SEND (FC 5)	The function block *AG_SEND* (FC 5) transfers the data from data type *DB1.UDT_1* to the communication partner. The inputs *ID* and *LADDR* must be appropriately configured for communication with the communication partner.
Data block *DB1*	The data type *UDT_1* is defined in the data block DB1.
Data type *UDT_1*	The addresses and types of the variables in SIMATIC 300 must be identical with the offsets and types of the controller. The data type *UDT_1* accepts the received user data and stores them until they are transmitted to the communication partner.

Table 205: Siemens SIMATIC 300 Configuration

▪ S&R TCP Configuration of the Siemens Controller SIMATIC 300

i The following step by step instructions for configuring the Siemens controller are not to be considered exhaustive.

This information is provided without guarantee (errors and omissions excepted); refer to the Siemens documentation when developing projects with Siemens controllers.

To create the S&R TCP server in the SIMATIC 300 project

1. Start the SIMATIC manager.
2. In the SIMATIC manager, open the project associated with the SIMATIC 300 controller.
3. In this project, create and configure the *Industrial Ethernet* and the *MPI* connections.

To create the UDT1 data type using the following variables

1. Open the *Function Blocks* folder in the Siemens SIMATIC manager.
2. Select **Add, S7 Block, Data Type** from the main menu and create a data type.
3. Name the data type **UDT1**
4. Give the symbolic name **UDT_1** to the data type.
5. In data type *UDT_1*, create the three **InOut_x** variables as described in the figure below.

Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	InOut_1	BYTE	B#16#0	
+1.0	InOut_2	BYTE	B#16#0	
+2.0	InOut_3	WORD	W#16#0	
=4.0		END_STRUCT		

Figure 55: List of Variables in the Siemens UDT1 Block

i During cyclic and acyclic data exchange, note that some controllers (e.g., SIMATIC 300) add so-called *pad bytes*. Pad bytes ensure that all data types greater than one byte always begin at an even offset and that also the total size of the defined variables is even.

In such a case, dummy bytes must be used on the correct place of the HIMax controller (see Chapter 8.6 Third-Party Systems with Pad Bytes).

To create the DB1 data block for the FC 5 and FC 6 function blocks

1. Select **Add, S7 Function Block, Data Block** on the main menu and create a data block.
2. Enter the name **DB1** for the data block.
3. Enter the symbolic name **DB1** for the data block.
4. Assign the *UDT_1* data type to the *DB1* data block.
5. In the *DB1* data block, configure the data types such as described in the figure below.

Adresse	Name	Typ	Anfangswert
0.0		STRUCT	
+0.0	Enable	BOOL	FALSE
+2.0	SendTime	S5TIME	S5T#100MS
+4.0	RecvTime	S5TIME	S5T#10MS
+6.0	UDT_1	"UDT_1"	
=10.0		END_STRUCT	

Figure 56: List of Variables in the Siemens DB1 Function Block

To create the following symbols in the Symbol Editor

1. Double click the **OB1** organization block to open the *KOP/AWL/FUP* dialog box.
2. Open the Symbol Editor on the main menu **Extras, Symbol Table.**
3. Add the variables **M 1.0...MW 5** in the *Symbol Editor* such as specified in the figure below.

S7-Programm(1) (Symbole) -- S7_Pro5\SIMATIC 300-Stat

	Status	Symbol	Adresse	Datentyp
1		Cycle Execution	OB 1	OB 1
2		DB1	DB 1	DB 1
3		UDT_1	UDT 1	UDT 1
4		RecDone	M 1.0	BOOL
5		RecError	M 1.1	BOOL
6		SendDone	M 1.2	BOOL
7		SendError	M 1.3	BOOL
8		RecStatus	MW 1	WORD
9		RecLen	MW 3	INT
10		SendStatus	MW 5	WORD
11				

Figure 57: SIMATIC Symbol Editor

To create the AG_RECV (FC 6) function block

1. Open the *KOP/AWL/FUP* dialog box.
2. From the structure tree located on the left side of the Symatic Manager, select the following function blocks in the given order:
   one OR gate
   one S_VIMP
   one AG_RECV (FC 6)
3. Drag these function blocks onto the *OB1* organization block.

4. Connect and configure the *function blocks* as described in the figure below.
5. Right-click the **AG_RECV (FC 6)** function block, and then click **Properties**.
6. Deactivate **Active Connection Setup** and configure the ports.
7. Note the *LADDR* function block parameter down and enter it in the function chart on the *AG_RECV (FC 6)* function block.

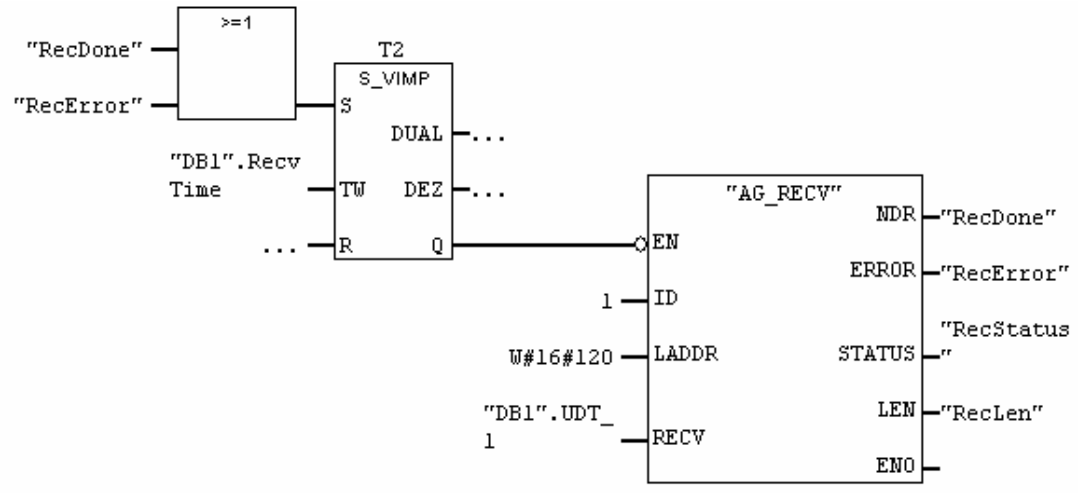


Figure 58: Receive Function Chart

To create the AG_SEND (FC 5) function block

1. Open the *KOP/AWL/FUP* dialog box.
2. From the structure tree located on the left side of the Symatic Manager, select the following function blocks in the given order:
   one OR gate
   one S_VIMP
   one AG_SEND (FC 5)
3. Drag these function blocks onto the *OB1* organization block.
4. Connect and configure the *function blocks* as described in the figure below.
5. Right-click the **AG_SEND (FC 5)** function block, and then click **Properties**.
6. Deactivate **Active Connection Setup** and configure the ports.
7. Note the *LADDR* function block parameter down and enter it in the function chart on the *AG_SEND (FC 5)* function block.

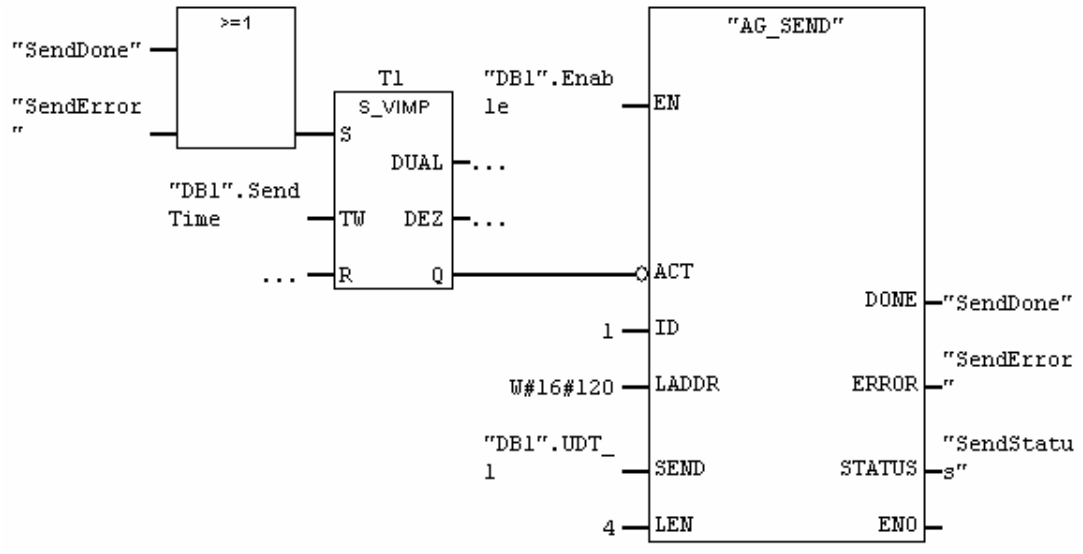


Figure 59: Send Function Chart

To load the code into the SIMATIC 300 controller

1. Start the **Code Generator** for the program.
2. Make sure that the code was generated without error.
3. Load the code into the SIMATIC 300 controller.

8.2.1 S&R TCP Configuration of the HIMax Controller

For more information on how to configure the HIMax controllers and use the SILworX programming tool, refer to the manual "First Steps in SILworX".

To create the following global variables in the Variable Editor

1. In the structure tree, open **Configuration, Global Variables.**
2. Right-click the **Global Variables**, and then click **Edit**.
3. Create the global variables as described in Table 206.

Name	Type
Siemens_HIMA1	Byte
Siemens_HIMA2	Byte
Siemens_HIMA3	WORD
HIMA_Siemens1	Byte
HIMA_Siemens2	Byte
HIMA_Siemens3	WORD

Table 206: Global Variables

To create the S&R TCP protocol in the resource

1. In the structure tree, open **Configuration, Resource**.
2. Right-click **Protocols,** then click **New**.
3. Select **Send/Receive over TCP** and enter a name for the protocol.
4. Click OK to create a new protocol.
5. Right-click **Send/Receive over TCP,** then click **Properties**.
6. Click **COM Module**. The remaining parameters retain the default values.

To create the TCP connection

1. Right-click **Send/Receive over TCP,** then click **New**.
2. Right-click **TCP Connection,** then click **Properties**.
3. Configure the properties such as specified in the figure.

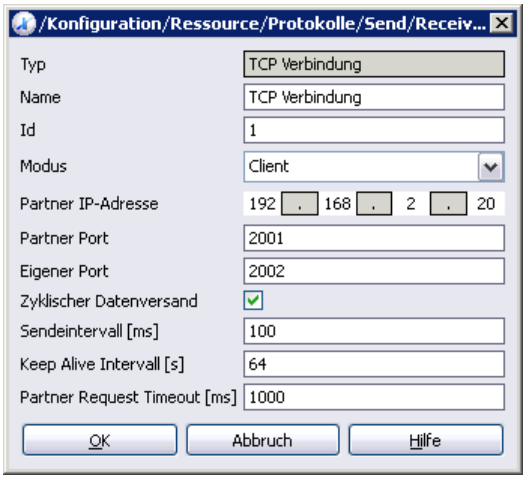


Figure 60: TCP Connection Properties in SILworX

i If parameters for cyclic data exchange between two controllers should be set, the option *Cyclic data transfer* must be activated in the *Properties* dialog box for the TCP Connection.

To configure the receive data of the HIMax controller

1. Right-click **TCP Connection,** then click **Edit**.
2. Select the **Process Variables** tab.
3. Drag the following global variables from the **Object Panel** onto the **Input Signals** area.

Global Variable	Type
Siemens_HIMA1	Byte
Siemens_HIMA2	Byte
Siemens_HIMA3	WORD

Table 207: Variables for Receive Data

4. Right-click anywhere in the **Register Inputs** area, and then click **New Offsets** to renumber the variable offsets.

i Take into account that the variable offsets in the HIMax controller must be identical with the variable addresses with *UDT_1* data type in the SIMATIC 300.

To configure the send data of the HIMax controller

1. Right-click **TCP Connection,** then click **Edit**.
2. Select the **Process Variables** tab.
3. Drag the following global variables from the **Object Panel** onto the **Input Signals** area.

Global Variable	Type
HIMA_Siemens1	Byte
HIMA_Siemens2	Byte
HIMA_Siemens3	WORD

Table 208: Variables for Send Data

4. Right-click anywhere in the **Register Inputs** area, and then click **New Offsets** to renumber the variable offsets.

i Take into account that the variable offsets in the HIMax controller must be identical with the variable addresses with *UDT_1* data type in the SIMATIC 300.

To verify the S&R TCP configuration

1. In the structure tree, open **Configuration, Resource, Protocols, Send/Receive Protocol over TCP**.
2. Click the **Verification** button on Action Bar, and then click **OK** to confirm the action.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

8.3 TCP S&R Protocols Menu Functions

8.3.1 Edit

The **Edit** dialog box for the S&R TCP protocol contains the following tab:

System Variables

The *system variables* are used to evaluate the state of the TCP Send Receive Protocol from within the user program.

Element	Description
Active Connection Count	System variable providing the number of active (not disturbed) connections.
Disturbed Connection Count	System variable providing the number of disturbed connections. Disturbed means that the TCP connection was interrupted due to a timeout or an error.
Status	No function

Table 209: System Variables S&R TCP

8.3.2 Properties

Over a TCP connection, data is exchanged cyclically or acyclically. The S&R TCP function blocks are required for the acyclic data exchange.

On a connection, data cannot be simultaneously exchanged cyclically and acyclically.

General

Name	Description
Type	Send/Receive over TCP
Name	Name for the current Send/Receive over TCP Protocol. A maximum of 31 characters.
Module	Selection of the COM module within which the protocol is processed.
Use Max CPU Load	Activated: Use CPU load limit from the Max. CPU Load [%] field. Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%

Table 210: S&R TCP General Properties

CPU/COM

The default values of the parameters provide the fastest possible data exchange of S&R TCP data between the COM module (COM) and the processor module (CPU) within the controller. These parameters should only be changed if it is necessary to reduce the COM and CPU loads for an application, and the process allows this change.

i Only experienced programmers should modify the parameters. Increasing the COM and CPU refresh rate means that the effective refresh rate of the S&R TCP data is also increased. The system time requirements must be verified.

Name	Description
Refresh Rate [ms]	Refresh rate in milliseconds at which the COM and CPU exchange S&R TCP protocol data. If the Refresh Rate is zero or lower than the cycle time for the controller, data is exchanged as fast as possible. Range of values: 0...(2^{31}-1), Default value: 0
Within one cycle	Activated: Transfer of the S&R TCP data from the CPU to the COM within a CPU cycle. Deactivated: Transfer of the S&R TCP data from the CPU to the COM, distributed over multiple CPU cycles, each with 1100 bytes per data direction.

Table 211: Parameters of COM/CPU

8.4 Menu Functions for TCP Connection

8.4.1 Edit

The **Edit** menu function opens the tabs Process Variables and System Variables.

Process Variables

Input Signals

The *Input Signals* area contains the variables for cyclic data exchange that this controller should receive.

Any variables can be created in the *Input Signals* tab. Offsets and types of the received variables must be identical with offsets and types of the transmitted variables (send data) of the communication partner.

Output Signals

The variables for cyclic data exchange sent by this controller are entered in the *Output Signals* area.

Any variables can be created in the *Output Signals* tab. Offsets and types of the received variables must be identical with offsets and types of the transmitted variables (receive data) of the communication partner.

8.4.2 System Variables

Using the variables in the *System Variables* tab, the state of the TCP connection can be assessed from within the user program.

Name	Description
Bytes received	Number of bytes received so far.
Bytes sent	Number of bytes sent so far.
Error Code	Error code of the TCP connection. See Chapter 8.8.4.
Error Code Timestamp [ms]	Millisecond fraction of the timestamp. Time point when the error occurred.
Error Code Timestamp [s]	Second fraction of the timestamp. Time point when the error occurred.
Partner Request Timeout	With acyclic data transfer: Timeout within which the communication partner must receive at least one time data after data sending. 0=Off 1 ... 2^{32}-1 [ms]
Partner Connection State	If no data is received within the timeout, 'Partner connection state' is set to *not connected* and the connection is restarted. 0=No connection 1=Connection OK
Status	TCP connection status. (see Chapter 8.8.6)

Table 212: System Variables

8.4.3 Properties

Over a TCP connection, data is exchanged cyclically or acyclically. The S&R TCP function blocks are required for the acyclic data exchange. The S&R TCP function blocks cannot be used for cyclic data exchange.

Name	Description
Type	TCP connection
Name	Any unique name for one TCP connection. A maximum of 31 characters.
ID	Any unique identification number (ID) for each TCP connection. The ID is also required as a reference for the S&R TCP function blocks. Range of values: 0..255 Default value: 0
Mode	Server (default value): This station operates as a server (passive mode). The connection is established by the communication partner (client). Once the connection has been established, both communication partners are equal and can send data at any time. The own port must be specified.
	Server with defined partner: This station operates as a server (passive mode). The connection is established by the communication partner (client). Once the connection has been established, both communication partners are equal and can send data at any time. If the IP address and/or port of the communication partner are defined here, only the specified communication partner can connect to the server. All other stations are ignored. If one of the parameters (IP address or port) is set to zero, the parameter is not verified.
	Client: This station operates as a client, i.e., the station establishes the connection to the communication partner. IP address and port of the communication partner must be specified. Also an own port can optionally be defined.
Partner IP Address	IP address of the communication partner. 0.0.0.0: any IP address is permitted. Valid range: 1.0.0.0 ... 223.255.255.255, except for: 127.x.x.x Default value: 0
Partner port	Port of the communication partner. Zero: Any port Ports that are reserved or already used (1...1024), are rejected by the COM OS. Range of values: 0..65535 Default value: 0

Own Port	Own port. Zero means any port. Ports that are reserved or already used (1 up to 1024), are rejected by the COM OS. Range of values: 0..65535 Default value: 0
Cyclic data transfer	Deactivated (default value) Cyclic data transfer is deactivated. Function blocks must be used to program the data exchange over this TCP connection. No cyclic E/A data may be defined on this connection.
	Activated: Cyclic data transfer is active. Data is defined in the Process Variable dialog box for the TCP connection. Receive data must be defined. No function blocks can be used on this connection.
Send Interval [ms]	Only editable with cyclic data transfer. The send interval is set here. Range of values 10...2147483647 ms (lower values are rounded to 10 ms) Default value: 0
Keep Alive Interval [s]	Time period until the connection monitoring provided by the TCP is activated. Zero deactivates the connection monitoring. If no data is exchanged within the specified KeepAlive interval, the KeepAlive samples are sent to the communication partner. If the connection still exists, the KeepAlive samples are acknowledged by the communication partner. If no data is exchanged between the partners within a period of > 10 KeepAlive interval, the connection is closed. If no response is received after a data packet sending, the data packet is resent in predefined intervals. The connection is aborted after 12 unsuccessful resends (approx. 7 minutes). Range of values 1...65535s Default value: 0 = deactivated
Partner Request Timeout [ms]	With acyclic data transfer: Timeout within which the communication partner must receive at least one time data after data sending. If no data is received within the timeout, *Partner connection state* is set to *not connected* and the connection is restarted. After a timeout or another error closed the connection, the active side re-establishes the connection with a delay of 10 x PartnerRequestTimeout or a delay of 10 seconds if PartnerRequestTimeout is equal to 0. The passive side opens the port within half of this time. 0=Off Range of values 1...2^{32}-1 [ms] Default value: 0

Table 213: S&R TCP Connection Properties

8.5 Data Exchange

S&R TCP operates according to the client/server principle. The connection is established by the communication partner which is configured as a client. Once the connection has

been established, both communication partners are equal and can send data at any point in time.

S&R TCP does not have its own data protection protocol; rather, it uses TCP/IP directly. As the data sent by the TCP are arranged in a "data stream", it must be ensured that offsets and types of the variables to be exchanged on the receiving station are identical with the ones on the sending station.

S&R TCP is compatible with the Siemens SEND/RECEIVE interface and allows cyclical data exchange with the Siemens S7 function blocks AG_SEND (FC5) and AG_RECV (FC6) (see Chapter 8.2, Example of S&R TCP Configuration).

Further, HIMA provides five S&R TCP function blocks for controlling and individually configuring communication using the user program. With the S&R TCP function blocks, any arbitrary protocol transferred over TCP (e.g., Modbus) can be sent and received.

8.5.1 TCP Connections

For each connection to a communication partner over S&R TCP, at least one TCP connection must exist in the HIMax controller.

The identification number of the TCP connection and the addresses/ports of the own controller and of the communication partner's controller must be set in the *Properties* dialog box for the TCP connection.

A maximum of 32 TCP connections can be established in a HIMax controller.

These TCP connections must have different identification numbers and different addresses/ports.

To create a new TCP connection

1. In the structure tree, open **Configuration, Resource**.
2. Right-click **Protocols,** then click **New**.
3. Select **Send/Receive over TCP** and enter a name for the protocol.
4. Click OK to create a new protocol.
5. Right-click **Send/Receive over TCP,** then click **Properties**.
6. Click **COM Module**. The remaining parameters retain the default values.

TIP The HIMax controller and the third-party system must be located in the same subnet or must have the corresponding routing settings if a router is used.

8.5.2 Cyclic Data Exchange

If data is exchanged cyclically, a send interval must be defined in the HIMax controller and in the communication partner.

The send interval defines the cyclic time period within which the sending communication partner sends the variables to the receiving communication partner.

- To ensure a continuous data exchange, both communication partners should define almost the same send interval (see Chapter 8.5.5, Flow Control).
- For cyclic data exchange, the *Cyclic Data Transfer* option must be activated in the TCP connection in use.
- If the *Cyclic Data Transfer* option is activated in a TCP connection, no function blocks may be used.
- The variables to be sent and received are assigned in the *Process Variable* dialog box for the TCP connection. Receive data **must** exist, send data is optional.

i The same variables (same offsets and types) that are defined as send data in a station, must be defined as receive data in the other station.

8.5.3 Acyclic Data Exchange with Function Blocks

In HIMax controllers, the acyclic data exchange is controlled by the user program over the TCP S&R function blocks.
Data exchange can thus be controlled using a timer or a mechanical switch connected to a physical input of the HIMax controller.

- The *Cyclic Data Transfer* option must be deactivated in the TCP connection in use.
- Only one S&R TCP function block may send at any given time.
- The variables to be sent or received are assigned in the *Process Variables* dialog box for the S&R TCP function blocks (all except for *Reset*).

i The same variables (same offsets and types) that are defined as send data in a station, must be defined as receive data in the other station.

8.5.4 Simultaneous Cyclic and Acyclic Data Exchange

For this purpose, one TCP connection must be configured for cyclic data and one TCP connection for acyclic data. The two TCP connections must use different partner IP addresses and partner ports.

One individual TCP connection cannot be simultaneously used for cyclic and acyclic data exchange.

8.5.5 Flow Control

The flow control is a component of the TCP and monitors the continuous data traffic between two communication partners.

With cyclic data transfer, at least one packet must be received after a maximum of 3 to 5 packets have been sent; otherwise, transmission is blocked until a packet is received or the connection monitoring process terminates the connection.

The number (3...5) of potential transmissions without packet reception depends on the size of the packets to be sent.

Number=5 for small packets < 4kB.

Number=3 for big packets ≥ 4kB.

- While planning the project, it must be ensured that no station sends more data than the other station can simultaneously process.
- To ensure a cyclical data exchange, both communication partners should define almost the same send interval

8.6 Third-Party Systems with Pad Bytes

During cyclic and acyclic data exchange, note that some controllers (e.g., SIMATIC 300) add so-called *pad bytes*. Pad bytes ensure that all data types exceeding one byte always begin at an even offset and that the total size of the packets (in bytes) is also even.

In the HIMax controller, *dummy bytes* must be added in place of *pad bytes* in the corresponding positions.

Address	Name	Type	Initial value
0.0		STRUCT	
+0.0	InOut_1	BYTE	B#16#0
+2.0	InOut_3	WORD	W#16#0
=4.0		END_STRUCT	

Figure 61: Siemens List of Variables

In the Siemens controller, a *pad byte* is added (not visible) such that the *InOut_3* variable begins at an even offset.

Ausgangssignale

F	Name	Datentyp	ffse	Transfer-Operation	Globale Variable
1	InOut_1	BYTE	0	5	InOut_1
2	Dummy	BYTE	1	5	Dummy
3	InOut_3	WORD	2	5	InOut_3

Figure 62: HIMax List of Variables

In the HIMax controller, a *dummy byte* must be added such that the *InOut_3* variable has the same offset as in the Siemens controller.

8.7 S&R TCP Function Blocks

If the cyclic data transfer is not sufficient flexible, data can also be sent and received using the S&R TCP function blocks. The *Cyclic Data Transfer* option must be deactivated in the TCP connection in use.

The S&R TCP function blocks is used to tailor the data transfer over TCP/IP to best meet the project requirements.

The function blocks are configured in the user program. The functions (Send, Receive, Reset) of the HIMatrix controller can thus be set and evaluated in the user program.

S&R TCP function blocks are required for the acyclic data exchange. These function blocks are not required for the cyclic data exchange between server and client.

i The configuration of the S&R TCP function blocks is described in Chapter 12.

The following function blocks are available:

Function block	Function Description
TCP_Reset (see Chapter 8.7.1)	TCP connection reset
TCP_Send (see Chapter 8.7.2)	Sending of data
TCP_Receive (see Chapter 8.7.3)	Reception of data packets with fixed length
TCP_ReceiveLine (see Chapter 8.7.4)	Reception of an ASCII line
TCP_ReceiveVar (see Chapter 8.7.5)	Reception of data packets with variable length (with length field)
LATCH	Only used within other function blocks
PIG	Only used within other function blocks
PIGII	Only used within other function blocks

Table 214: Function Blocks for S&R TCP Connections

8.7.1 TCP_Reset

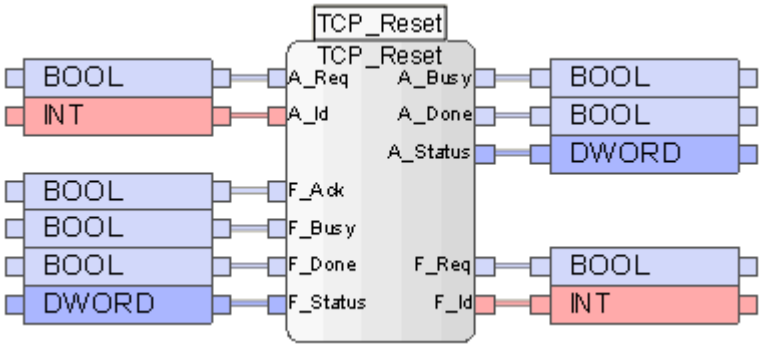


Figure 63: Function Block TCP_Reset

The **TCP Reset** function block is used to re-establish a disturbed connection if a send or receive function block reports a timeout error (16#8A).

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A-Inputs	Description	Type
A_Req	Rising edge starts the function block	BOOL
A_Id	Identification number (*ID*) of the disturbed TCP connection to be reset.	INT

Table 215: A-Inputs for the TCP_Reset Function Block

A_Outputs	Description	Type
A_Busy	TRUE: The TCP connection is still being reset.	BOOL
DONE	TRUE: The data transmission ended without error.	BOOL
A_Status	The status and error code of the function block and of the TCP connection are output on *A_Status*.	DWORD

Table 216: A-Outputs for the TCP_Reset Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **Reset** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **Reset** function block (in the Function Blocks folder) to the **TCP_Reset** function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **TCP_Reset** function block in the user program to the same variables that will be connected to the outputs of the **Reset** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Busy	BOOL
F_Done	BOOL
F_Status	DWORD

Table 217: F-Inputs for the TCP_Reset Function Block

Connect the *F-Outputs* of the **TCP_Reset** function block in the user program to the same variables that will be connected to the inputs of the **Reset** function block in the structure tree.

F-Outputs	Type
F_Req	BOOL
F_Id	DWORD

Table 218: F-Outputs for the TCP_Reset Function Block

To create the Reset function block in the structure tree:

1. In the structure tree, open **Configuration, Resource, Protocols, Send/Receive over TCP, Function Blocks, New.**
2. Select the **Reset** function block and click **OK**.
3. Right-click the **Reset** function block, and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **Reset** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **TCP_Reset** function block in the user program.

Inputs	Type
ID	DWORD
REQ	BOOL

Table 219: Input System Variables

Connect the following outputs of the **Reset** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **TCP_Reset** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
DONE	BOOL
STATUS	DWORD

Table 220: Output System Variables

To operate the TCP_Reset function block, the following steps are essential:

1. In the user program, set the identification number for the disturbed TCP connection on the *A_ID* input.
2. In the user program, set the *A_Req* input to TRUE.

i The function block reacts to a rising edge on *A_Req*.

The *A_Busy* output is set to TRUE until a reset is sent to the specified TCP connection. Afterwards, *A_Busy* is set to FALSE and *A_Done* is set to TRUE.

8.7.2 TCP_Send

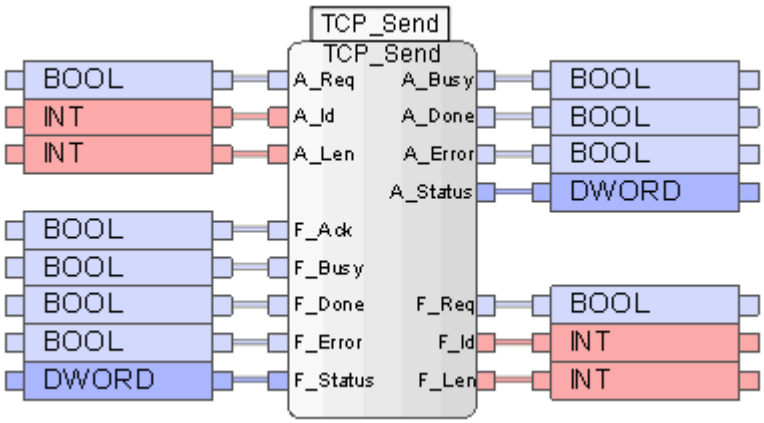


Figure 64: Function Block TCP_Send

The **TCP_Send** function block is used for acyclically send variables to a communication partner. A function block with the same variables and offsets, e.g., *Receive,* must be configured in the communication partner.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A-Inputs	Description	Type
A_Req	The rising edge starts the function block.	BOOL
A_Id	Identification number of the configured TCP connection to the communication partner to which data should be sent.	INT
A_Len	Number of transmitted variables, expressed in bytes. A_Len must be greater than zero and must not end within a variable.	INT

Table 221: A-Inputs for the TCP_Send Function Block

A_Outputs	Description	Type
A_Busy	TRUE: Data is still being transmitted.	BOOL
DONE	TRUE: The data transmission ended without error.	BOOL
ERROR	TRUE: An error occurred FALSE: No error	BOOL
A_Status	The status and error code of the function block and of the TCP connection are output on *A_Status*.	DWORD

Table 222: A-Outputs for the TCP_Send Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **Send** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **Send** function block (in the Function Blocks folder) to the **TCP_Send** function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **TCP_Send** function block in the user program to the same variables that will be connected to the outputs of the **Send** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Busy	BOOL
F_Done	BOOL
F_Error	BOOL
F_Status	DWORD

Table 223: F-Inputs for the TCP_Send Function Block

Connect the *F-Outputs* of the **TCP_Send** function block in the user program to the same variables that will be connected to the inputs of the **Send** function block in the structure tree.

F-Outputs	Type
F_Id	DWORD
F_Len	INT
F_Req	BOOL

Table 224: F-Outputs for the TCP_Send Function Block

To create the Send function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, Send/Receive over TCP, Function Blocks, New.**
2. Select the **Send** function block and click **OK**.
3. Right-click the **Send** function block, and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **Send** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **TCP_Send** function block in the user program.

Inputs	Type
ID	DWORD
LEN	INT
REQ	BOOL

Table 225: Input System Variables

Connect the following outputs of the **Send** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **TCP_Send** function block in the user program.

Outputs	Type
Ack	BOOL
Busy	BOOL
Done	BOOL
ERROR	BOOL
STATUS	DWORD

Table 226: Output System Variables

Data	Description
Send Data	Any variables can be created in the *Process Variables* tab. Offsets and types of the received variables must be identical with offsets and types of the transmitted variables of the communication partner.

Table 227: Send Data

To operate the TCP_Send function block, the following steps are essential:

i The send variables must be created in the *Process Variables* tab of the *Send* dialog box. Offsets and types of the received variables must be identical with offsets and types of the transmitted variables of the communication partner.

1. In the user program, set the identification number of the TCP connection on the *A_ID* input.
2. In the user program, set the expected length (in bytes) of the variables to be sent on the *A_Len* input.
3. In the user program, set the *A_Req* input to TRUE.

i The function block reacts to a rising edge on *A_Req*.

The A_Busy output is set to TRUE until the variables have been sent. Afterwards, A_Busy is set to FALSE and A_Done is set to TRUE.

If the sending process was not successful, the A_Error output is set to TRUE and an error code is output on A_Status.

8.7.3 TCP_Receive

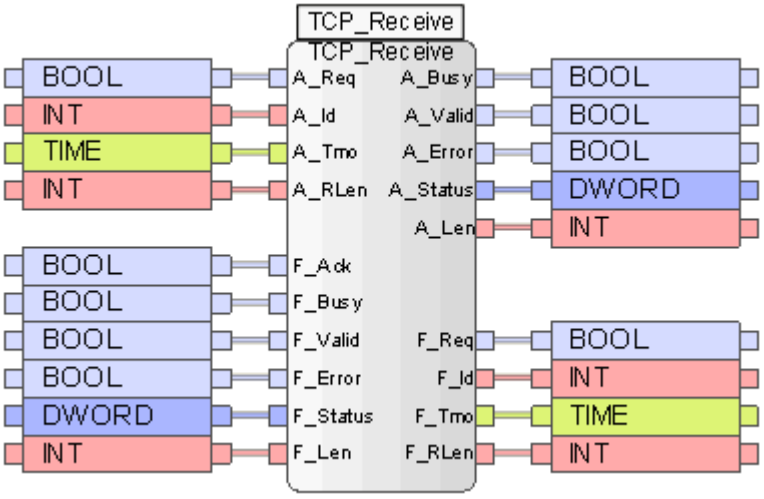


Figure 65: Function Block TCP_Receive

The **TCP_Receive** function block is used to receive predefined variables from the communication partner.

A function block with the same variables and offsets, e.g., TCP_Send, must be configured in the communication partner.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A-Inputs	Description	Type
A_Req	The rising edge starts the function block.	BOOL
A_Id	Identification number of the configured TCP connection to the communication partner from which data should be received.	INT
A_Tmo	Receive timeout If no data are received within the timeout, the function block stops and an error message appears. If the *A_Tmo* input is not used or set to zero, the timeout is deactivated.	TIME
A_RLen	*A_RLen* is the expected length of the variables to be received, expressed in bytes. *A_RLen* must be greater than zero and must not end within a variable.	INT

Table 228: A-Inputs for the TCP_Receive Function Block

A_Outputs	Description	Type
A_Busy	TRUE: Data is still being received.	BOOL
A_Valid	TRUE: The data reception ended without error.	BOOL
ERROR	TRUE: An error occurred FALSE: No error	BOOL
A_Status	The status and error code of the function block and of the TCP connection are output on *A_Status*.	DWORD
A_Len	Number of received bytes.	INT

Table 229: A-Outputs for the TCP_Receive Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **Receive** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **Receive** function block (in the Function Blocks folder) to the **TCP_Receive** function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **TCP_Receive** function block in the user program to the same variables that will be connected to the outputs of the **Receive** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Busy	BOOL
F_Valid	BOOL
F_Error	BOOL
F_Status	DWORD
F_Len	INT

Table 230: A-Inputs for the TCP_Receive Function Block

Connect the *F-Outputs* of the **TCP_Receive** function block in the user program to the same variables that will be connected to the inputs of the **Receive** function block in the structure tree.

F-Outputs	Type
F_Req	BOOL
F_Id	DWORD
F_Tmo	INT
F_RLen	INT

Table 231: F-Outputs for the TCP_Receive Function Block

To create the corresponding Receive function block in the structure tree:

1. In the structure tree, open **Configuration, Resource, Protocols, Send/Receive over TCP, Function Blocks, New.**
2. Select the **Receive** function block and click **OK**.
3. Right-click the **Receive** function block, and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **Receive** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **TCP_Receive** function block in the user program.

Inputs	Type
ID	INT
REQ	BOOL
RLEN	INT
TIMEOUT	TIME

Table 232: Input System Variables

Connect the following outputs of the **Receive** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **TCP_Receive** function block in the user program.

Outputs	Type
Ack	BOOL
Busy	BOOL
ERROR	BOOL
LEN	INT
STATUS	DWORD
VALID	BOOL

Table 233: Output System Variables

Data	Description
Receive variables	Any variables can be created in the *Process Variables* tab. Offsets and types of the received variables must be identical with offsets and types of the transmitted variables of the communication partner.

Table 234: Receive Variables

To operate the TCP_Receive function block, the following steps are essential:

i The receive variables must be created in the *Process Variables* tab located in the *Receive* dialog box. Offsets and types of the receive variables must be identical with offsets and types of the send variables of the communication partner.

1. In the user program, set the identification number for the TCP connection on the *A_ID* input.
2. In the user program, set the receive timeout on the *A_Tmo* input.
3. In the user program, set the expected length of the variables to be received on the *A_RLen* input.
4. In the user program, set the *A_Req* input to TRUE.

i The function block starts with a rising edge on *A_Req*.

The *A_Busy* output is set to TRUE until the variables have been received or the receive timeout has expired. Afterwards, *A_Busy* is set to FALSE and *A_Valid* or *A_Error* to TRUE.

If no error occurred during the variable reception, the *A_Valid* output is set to TRUE. The variables defined in the *Data* tab can be evaluated.

If an error occurred during the variable reception, the *A_Error* output is set to TRUE and an error is output on *A_Status*.

8.7.4 TCP_ReceiveLine

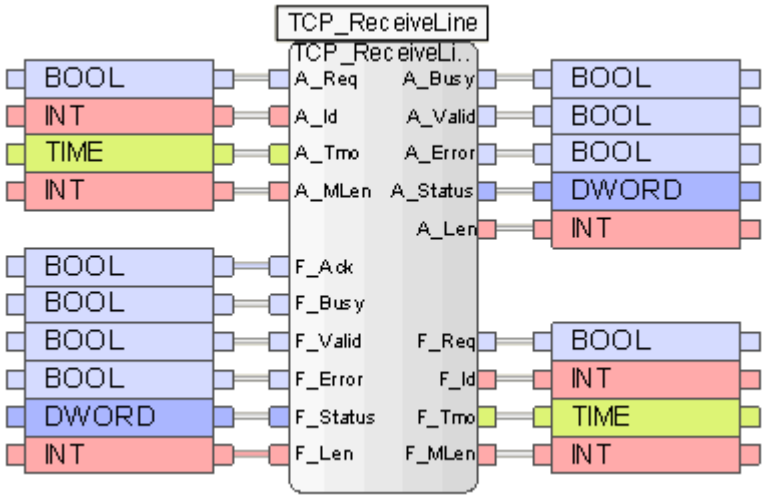


Figure 66: Function Block TCP_ReceiveLine

The **TCP_ReceiveLine** function block is used for receiving an ASCII character string with LineFeed (16#0A) from a communication partner.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Inputs and Outputs of the Function Block with Prefix A:

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A-Inputs	Description	Type
A_Req	Rising edge starts the function block.	BOOL
A_Id	Identification number of the configured TCP connection to the communication partner from which data should be received.	INT
A_Tmo	Receive timeout If no data are received within the timeout, the function block stops and an error message appears. If the input is not used or set to zero, the timeout is deactivated.	TIME
A_MLen	Maximum length of a line to be received, expressed in bytes. The receive variables must created in the *Data* tab located in the COM function block. Transmitted bytes = Min (A_MLen, line length, length of the data range).	INT

Table 235: A-Inputs for the TCP_ReceiveLine Function Block

A_Outputs	Description	Type
A_Busy	TRUE: Data is still being received.	BOOL
A_Valid	TRUE: The data reception ended without error.	BOOL
ERROR	TRUE: An error occurred FALSE: No error	BOOL
A_Status	The status and error code of the function block and of the TCP connection are output on A_Status.	DWORD
A_Len	Number of received bytes.	INT

Table 236: A-Outputs for the TCP_ReceiveLine Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **ReceiveLine** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **ReceiveLine** function block in the structure tree (located in the Function Blocks folder) to the **TCP_ReceiveLine** function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **TCP_ReceiveLine** function block in the user program to the same variables that will be connected to the outputs of the **ReceiveLine** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Busy	BOOL
F_Valid	BOOL
F_Error	BOOL
F_Status	DWORD
F_Len	INT

Table 237: F-Inputs for the TCP_ReceiveLine Function Block

Connect the *F-Outputs* of the **TCP_ReceiveLine** function block in the user program to the same variables that will be connected to the inputs of the **ReceiveLine** function block in the structure tree.

F-Outputs	Type
A_Req	BOOL
A_Id	INT
A_Tmo	TIME
A_MLen	INT

Table 238: A-Outputs for the TCP_ReceiveLine Function Block

To create the corresponding ReceiveLine function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, Send/Receive over TCP, Function Blocks, New.**
2. Select the **ReceiveLine** function block and click **OK**.
3. Right-click the **ReceiveLine** function block, and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **ReceiveLine** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **TCP_ReceiveLine** function block in the user program.

Inputs	Type
ID	INT
MLEN	INT
REQ	BOOL
TIMEOUT	TIME

Table 239: Input System Variables

Connect the following outputs of the **ReceiveLine** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **TCP_ReceiveLine** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
ERROR	BOOL
LEN	INT
STATUS	DWORD
VALID	BOOL

Table 240: Output System Variables

Data	Description
Receive variables	The *Process Variables* tab should only contain variables of type BYTE. Offsets of the variables must be identical with offsets of the variables of the communication partner.

Table 241: Receive Variables

To operate the TCP_ReceiveLine function block, the following steps are essential:

The receive variables of type BYTE must be created in the tab *Process Variables* located in the *ReceiveLine* dialog box. Offsets of the receive variables must be identical with offsets of the send variables of the communication partner.

1. In the user program, set the identification number for the TCP connection on the *A_ID* input.
2. In the user program, set the receive timeout on the *A_Tmo* input.
3. In the user program, set the maximum length of the line to be received on the *A_MLen* input.

A_Mlen must be greater than zero and determines the size of the receive buffer in bytes.

If the receive buffer is full and a line end has not yet occurred, the reading process ends and no error message appears.

The number of received bytes is output on the *A_Len* output:

Received bytes = Min (A_MLen, line length, length of the data range).

4. In the user program, set the *A_Req* input to TRUE.

The function block reacts to a rising edge on *A_Req*.

The *A_Busy* output is set to TRUE if the receive buffer is full or the end of line *LineFeed* is received or the receive time-put has expired. Afterwards, *A_Busy* is set to FALSE and *A_Valid* or *A_Error* to TRUE.

If no error occurred during the line reception, the *A_Valid* output is set to TRUE. The variables defined in the Data tab can be evaluated.

If an error occurred during the line reception, the *A_Error* output is set to TRUE and an error is output on *A_Status*.

8.7.5 TCP_ReceiveVar

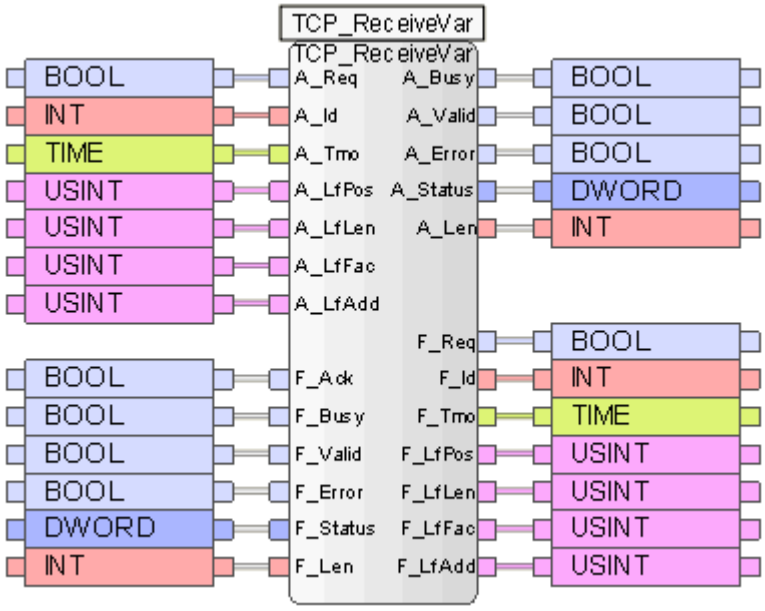


Figure 67: Function Block TCP_ReceiveVar

The **TCP_ReceiveVar** function block is used to evaluate data packets with variable length and containing a length field.

i To configure the function block, drag it from the function block library onto the user program (see also Chapter 12).

Function Description

The received data packets must have the structure represented in the figure below (e.g., Modbus protocol). Modifying the input parameters *A_LfPos, A_LfLen, A_LfFac, A_LfLen*, the received data packets can be adapt to any protocol format.

The received data packet is composed of a header and a data range. The header contains data such as participant address, telegram function, length field etc. required for establishing communication. To evaluate the data range, separate the header and read the length field.

The size of the header is entered in the *A_LfAdd* parameter.

The length of the data range must be read from the length field of the data packet currently read. The position of the length field is entered in *A_LfPos*. The size of the length field expressed in bytes is entered in *LfLen*. If the length is not expressed in bytes, the corresponding conversion factor must be entered in *A_LfFac* (e.g., 2 for WORD or 4 for DOUBLE WORD).

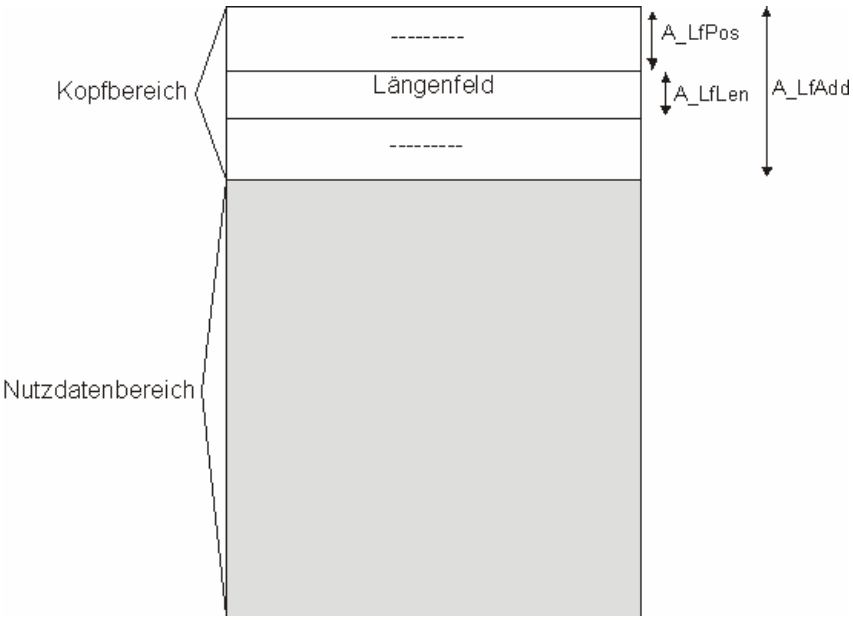


Figure 68: Data Packet Structure

Inputs and Outputs of the Function Block with Prefix A

These inputs and outputs can be used to control and evaluate the function blocks using the user program. The prefix "A" means "Application".

A-Inputs	Description	Type
A_Req	Rising edge starts the function block.	BOOL
A_Id	Identification number (*ID*) of the configured TCP connection to the communication partner from which the data should be received.	DWORD
A_Tmo	Receive timeout If no data are received within the timeout, the function block stops and an error message appears. If the input is not used or set to zero, the timeout is deactivated.	INT
A_LfPos	Start position of the length field in the data packet; the numbering starts with zero. (measured in bytes)	USINT
A_LfLen	Size of the *A_LfLen* length field in bytes. Permitted: 1, 2 or 4 bytes.	USINT
A_LfFac	Conversion factor in bytes if the value set in the length field is not expressed in bytes. If the input is not used or set to zero, **1** is used as default value.	USINT
A_LfAdd	Size of the header in bytes.	USINT

Table 242: A-Inputs for the TCP_ReceiveVar Function Block

A-Outputs	Description	Type
A_Busy	TRUE: Data is still being received.	BOOL
A_Valid	TRUE: The data reception ended without error.	BOOL
ERROR	TRUE: An error occurred during the reading process FALSE: No error	BOOL
A_Status	The status and error code of the function block and of the TCP connection are output on *A_Status*.	DWORD
A_Len	Number of received bytes.	INT

Table 243: A-Outputs for the TCP_ReceiveVar Function Block

Inputs and Outputs of the Function Block with Prefix F:

These inputs and outputs of the function block establish the connection to the **ReceiveVar** function block in structure tree. The prefix "F" means "Field".

i Common variables are used to connect the **ReceiveVar** function block in the structure tree (located in the Function Blocks folder) to the **TCP_ReceiveVar** function block (in the user program). These must be created beforehand using the Variable Editor.

Connect the *F-Inputs* of the **TCP_ReceiveVar** function block in the user program to the same variables that will be connected to the outputs of the **ReceiveVar** function block in the structure tree.

F-Inputs	Type
F_Ack	BOOL
F_Busy	BOOL
F_Valid	BOOL
F_Error	BOOL
A_Status	DWORD
A_Len	INT

Table 244: F-Inputs for the TCP_ReceiveVar Function Block

Connect the *F-Outputs* of the **TCP_ReceiveVar** function block in the user program to the same variables that will be connected to the inputs of the **ReceiveVar** function block in the structure tree.

F-Outputs	Type
F_Req	BOOL
F_Id	INT
F_Tmo	TIME
F_LfPos	USINT
A_LfLen	USINT
A_LfFac	USINT
A_LfAdd	USINT

Table 245: F-Outputs for the TCP_ReceiveVar Function Block

To create the ReceiveVar function block in the structure tree

1. In the structure tree, open **Configuration, Resource, Protocols, Send/Receive over TCP, Function Blocks, New.**
2. Select the **ReceiveVar** function block and click **OK**.
3. Right-click the **Receive** function block, and then click **Edit**.
   ☑ The window for assigning variables to the function blocks appears.

Connect the inputs of the **ReceiveVar** function block in the structure tree to the same variables that have been previously connected to the *F-Outputs* of the **TCP_ReceiveVar** function block in the user program.

Inputs	Type
ID	INT
Lf Add	USINT
Lf Fac	USINT
Lf Len	USINT
Lf Pos	USINT
REQ	BOOL
TIMEOUT	TIME

Table 246: Input System Variables

Connect the following outputs of the **ReceiveVar** function block in the structure tree to the same variables that have been previously connected to the *F-Inputs* of the **TCP_ReceiveVar** function block in the user program.

Outputs	Type
ACK	BOOL
BUSY	BOOL
ERROR	BOOL
LEN	INT
STATUS	DWORD
VALID	BOOL

Table 247: Output System Variables

Data	Description
Receive variables	Any variables can be created in the *Process Variables* tab. Offsets and types of the received variables must be identical with offsets and types of the transmitted variables of the communication partner.

Table 248: Receive Variables

To operate the TCP_ReceiveVar function block, the following steps are essential:

i The receive variables must be created in the *Process Variables* tab located in the *Variables* dialog box. Offsets and types of the receive variables must be identical with offsets and types of the send variables of the communication partner.

1. In the user program, set the identification number for the TCP connection on the *A_ID* input.
2. In the user program, set the receive timeout on the *A_Tmo* input.
3. In the user program, set the parameters A_LfPos, A_LfLen, A_LfFac and A_LfAdd.
4. In the user program, set the *A_Req* input to TRUE.

i The function block starts with a rising edge on *A_Req*.

The *A_Busy* output is set to TRUE until the variables have been received or the receive timeout has expired. Afterwards, *A_Busy* is set to FALSE and *A_Valid* or *A_Error* to TRUE.

If no error occurred during the variable reception, the *A_Valid* output is set to TRUE. The variables defined in the Data tab can be evaluated. The *A_Len* output contains the amount of data in bytes that was actually read.

If an error occurred during the variable reception, the *A_Error* output is set to TRUE and an error is output on *A_Status*.

8.8 Control Panel (Send/Receive over TCP)

The Control Panel can be used to verify and control the settings for the Send/Receive protocol. Details about the current status of the Send/Receive protocol (e.g., disturbed connections) are displayed.

To open Control Panel for monitoring the Send/Receive protocol

1. In the structure tree, click **Resource**.
2. Click **Online** on the **Action Bar**.
3. In the **System Log-in** window, enter the access data to open the Control Panel for the resource.
4. In the structure tree for the Control Panel, select **Send/Receive Protocol**.

8.8.1 Context Menu (Send/Receive Protocol)

The following command can be chosen from the context menu for the selected Send/Receive protocol:

Reset:

Reset the statistical data (e.g., min./max. cycle time etc.) to 0.

8.8.2 View Box (Send/Receive Protocol)

The view box displays the following values of the selected Send/Receive protocol.

Element	Description
Name	TCP SR Protocol
CPU Load (planned) [%]	See Chapter 7.4.2.
CPU Load (actual) [%]	
Undisturbed Connections	Disturbed Connection Count
Disturbed Connections	Disturbed Connection Count

Table 249: S&R Protocol View Box

8.8.3 View Box (Send/Receive Server)

The view box displays the following values of the selected Modbus client.

Element	Description
Name	Name of the Modbus slaves
Partner Timeout [ms]	Timeout within which the communication partner received at least one time data after data sending.
Connection State	Current state of this connection 0x00: Connection OK 0x01: Connection closed 0x02: Server waits for the connection to be established 0x04: Client attempts to establish connection 0x08: Connection is blocked
Peer Address	IP address of the communication partner.
Peer Port	Port of the communication partner.
Own Port	Port of this controller
Error Code	Error code (see Chapter 8.8.4)

Table 250: View Box of the Modbus Slave

8.8.4 Error Code of the TCP Connection

The error codes can be read from the *Error Code* variable.

For each configured connection: The connection state is composed of the connection state and error code of the last operation.

Error Code Decimal	Error Code Hexadecimal	Description
0	16#00	OK
4	16#04	Interrupted system call
5	16#05	I/O Error
6	16#06	Device unknown
9	16#09	Invalid socket descriptor
12	16#0C	No memory available
13	16#0D	Access Denied
14	16#0E	Invalid address
16	16#10	Device occupied
22	16#16	Invalid value (e.g., in the length field)
23	16#17	Descriptor table is full
32	16#20	Connection aborted
35	16#23	Operation is blocked
36	16#24	Operation currently in process
37	16#25	Operation already in process
38	16#27	Target address required
39	16#28	Message to long
40	16#29	Incorrect protocol type for the socket
42	16#2A	Protocol not available
43	16#2B	Protocol not supported
45	16#2D	Operation on socket not supported
47	16#2F	The address is not supported by the protocol
48	16#30	Address already in use
49	16#31	The address cannot be assigned
50	16#32	Network is down
53	16#35	Software caused connection abort
54	16#36	Connection reset by peer
55	16#37	No buffer space available
56	16#38	Socket already connected
57	16#39	Socket not connected
58	16#3A	Socket closed
60	16#3C	Operation time expired
61	16#3D	Connection refused (from peer)
65	16#41	No route to peer host
78	16#4E	Function not available
254	16#FE	Timeout occurred
255	16#FF	Connection closed by peer

Table 251: Error Codes of the TCP Connection

8.8.5 Additional Error Code Table for the Function Blocks

The error codes for the function blocks (e.g., 16#8x) are only output on A_Status of the S&R TCP function blocks.

Error Code Decimal	Error Code Hexadecimal	Description
129	16#81	Unknown connection ID
130	16#82	Invalid length
131	16#83	Only cyclic data is permitted for this connection
132	16#84	Connection is not currently available
133	16#85	Timeout value too large
134	16#86	Internal program error
135	16#87	Configuration error
136	16#88	Transmitted data does not correspond to the configured data structure
137	16#89	Function block stopped
138	16#8A	Timeout occurred or transmission blocked
139	16#8B	Another function block of this type is already active on this connection

Table 252: Additional Error Codes

8.8.6 Connection State

Error Code Decimal	Error Code Hexadecimal	Description
0	16#00	Connection OK
1	16#01	Connection closed
2	16#02	Server waits for the connection to be established
4	16#04	Client attempts to establish connection
8	16#08	Connection is blocked

Table 253: Connection State

8.8.7 Partner Connection State

Protocol State Decimal	Description
0	No connection
1	Connection OK

Table 254: Partner's Connection State

9 SNTP Protocol

(Simple Network Time Protocol)

The SNTP protocol is used to synchronize the time of the SNTP client over the SNTP server.

HIMax controllers can be configured and used as **SNTP server** and as **SNTP client**. The SNTP standard in accordance with RFC 2030 (SNTP version 4) applies with the limitation that only the unicast mode is supported.

Equipment and system requirements

Element	Description
Controller	HIMax with COM module or only with processor module
Activation	This function is activated by default in all HIMax systems.
Interface	Ethernet 10/100/1000BaseT

Table 255: Equipment and System Requirements for the S&R TCP

9.1 SNTP Client

To synchronize time settings, the SNTP client always uses the available SNTP server with the highest priority.

One SNTP client can be configured for time synchronization in each resource.

To create a new SNTP client

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. Right-click **Protocols**, then click **New**, **SNTP Client.**
   ☑ A new SNTP Client is created.
3. Right-click the SNTP client, and click **Properties** and select the **COM module**.

The dialog box for the SNTP client contains the following parameters:

Element	Description
Type	SNTP Client
Name	Name for the SNTP client, composed of a maximum of 32 characters.
Module	Selection of the COM or processor module within which the protocol is processed.
Use Max CPU Load	Activated: Use CPU load limit from the Max. CPU Load [%] field. Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%
Description	Any unique description for the SNTP client

Current SNTP Version	The current SNTP version is displayed.
Reference Stratum	The stratum of an SNTP client specifies the precision of its local time. The lowest the stratum, the more precise its local time. Zero means an unspecified or not available stratum (invalid). The SNTP server currently used by an SNTP client is the one that can be reached and has the highest priority. If the stratum of the current SNTP server is lower than the stratum of the SNTP client, the resource adopts the time of the current SNTP server.
Reference Stratum (continuation)	If the stratum of the current SNTP server is higher than the stratum of the SNTP client, the resource does not adopt the time of the current SNTP server. If the stratum of the current SNTP server is identical with the stratum of the SNTP client, two different cases result: ▪ If the SNTP Client (resource) only operates as SNTP client, the resource adopts the time of the current SNTP server. ▪ If the SNTP client (resource) also operates as SNTP server, the resource adopts the half value of the time difference to the current SNTP server per SNTP client request (time approaches slowly). Range of values: 1...16 Default value: 15
Client Time Request Interval [s]	Time interval within which the current SNTP Server performs the time synchronization. The Client Request Time Interval set in the SNTP client must be higher than the timeout set in the SNTP server. Range of values: 16...16384 s Default value: 16

Table 256: SNTP Client Properties

9.2 SNTP Client (Server Information)

The connection to the SNTP server is configured in the SNTP Server Info.

1 to 4 SNTP Server Info can be subordinate to a SNTP client.

To create a new SNTP Server Info

1. In the structure tree, open **Configuration, Resource, Protocols, SNTP Client**.
2. Right-click **SNTP Client,** and then click **New**, **SNTP Server Info**.
   ☑ A new SNTP Server Info is created.
3. Right-click the SNTP-Server Info, click **Properties , and then click the COM module**.

The dialog box for the SNTP Server Info contains the following parameters:

Element	Description
Type	SNTP Server Info
Name	Name for the SNTP server. A maximum of 32 characters.
Description	Description for the SNTP server. A maximum of 31 characters.
IP Address	IP address of the resource or PC in which the SNTP Server is configured. Default value: 0.0.0.0
SNTP Server Priority	Priority with which the SNTP client addresses this SNTP server. The SNTP servers configured for the SNTP client should have different priorities. Range of values: 0 (lowest priority) to 4294967295 (highest priority). Default value: 1
SNTP Server Timeout[s]	The timeout in the SNTP server must be set lower than the value for the *Time Request Interval* in the SNTP client. Range of values: 1...16384s Default value: 1

Table 257: SNTP Server Info Properties

9.3 SNTP Server

The SNTP server accepts the requests from a SNTP client and sends back to the SNTP client its current time.

Time synchronization of a remote I/O performed by a HIMax controller.
An SNTP server must be set up on the HIMax communication module connected to the remote I/O.

To create a new SNTP server

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. Right-click **Protocols,** and then click **New**, **SNTP Server**.
   ☑ A new SNTP Server is created.
3. Right-click the SNTP Server, click **Properties , and then click the COM module**...

The dialog box for the SNTP server contains the following parameters:

Element	Description
Type	SNTP Server
Name	Name for the SNTP server, composed of a maximum of 31 characters.
Module	Selection of the COM or processor module within which the protocol is processed.
Use Max CPU Load	Activated: Use CPU load limit from the Max. CPU Load [%] field. Deactivated: Do not use the CPU Load limit for this protocol.
Max. CPU Load [%]	Maximum CPU load of the module that can be used for processing the protocols. Range of values: 1...100% Default value: 30%
Description	Description for the SNTP
Current SNTP Version	The current SNTP version is displayed.
Stratum of Timeserver	The stratum of an SNTP server specifies the precision of its local time. The lowest the stratum, the more precise the local time. Zero means an unspecified or not available stratum (invalid). The value for the SNTP server stratum must be lower or equal to the stratum value of the requesting SNTP client. Otherwise, the SNTP client does not accept the SNTP server time. Range of values: 1 ... 15 Default value: 14

Table 258: SNTP Server Properties

10 X OPC Server

The HIMA X-OPC Server serves as transmission interface between HIMax controllers and third-party systems that are equipped with an OPC interface.

OPC means Openess, Productivity & Collaboration and is based on the technology developed by Microsoft. This technology allows the user to interconnect process control systems, visualization systems and controllers from different manufacturers, and it enables them to exchange data with one another (see also www.opcfoundation.org). After installation, the HIMA X OPC server is run on a PC as Windows service.

i SILworX is used to configure and operate the entire X OPC server. The X OPC server can be loaded, started and stopped in the SILworX Control Panel like a controller.

The X OPC server supports the following specifications:

- **Data Access (DA) versions 1.0, 2.05a and 3.0**
  DA is used to ensure the process data transmission from the HIMax controller to the OPC client.
  Each global variable of the HIMax controller can be transferred to a OPC client.
- **Alarm&Event (A&E) version 1.10**
  A&E is used to transfer alarms and events from HIMax controller to the OPC client.
  Each global variable of the HIMax controller can be monitored using sequence of events recording.
  Events are state modifications of a variable that are performed by the plant or controllers and are provided with a timestamp.
  Alarms are events that signalize an increasing risk potential.
  Events are divided in Boolean and scalar events, see Chapter 10.7.

10.1 Equipment and System Requirements

Element	Description
Activation	Software activation code required, see Chapter 3.5. The following licenses can be activated on an individual basis: ▪ Data Access (DA) Server ▪ Alarm and Events (A&E) Server
PC Operating System	The OPC server can run on an x86 based PC with the following operating systems: ▪ Windows XP Professional (mind. Service Pack 2) (32-bit ▪ Windows Server 2003 (32-bit) ▪ Windows Vista Ultimate (32-bit) ▪ Windows Vista Business (32-bit)
Requirements to the Host PC	Minimum requirements to the host PC: ▪ Pentium 4 ▪ 1 GByte (XP) or 2.5 GByte RAM (Vista) ▪ The network card must be designed according to the data traffic (100 Mbit/s or 1 Gbit/s). i The minimum requirements only apply to the operation of a X OPC server if no additional applications, such as SILworX or Word, is run on the host PC.

Table 259: Equipment and System Requirements for the X-OPC Server

10.2 X-OPC Server Properties

Element	Description
OPC Server	The X OPC server supports the following functions: ▪ OPC Data Access Custom Interface, versions 1.0, 2.05a and 3.0. ▪ OPC Alarm & Event Interfaces 1.10
Safety-related	The X-OPC server is run on a PC and is not safety-related.
Interface	Recommended: Ethernet 1GBit/s
Data Exchange	Data exchange via safe**ethernet**.
Ethernet Network	The underlying Ethernet network speed must be designed according to the data traffic (min. 100 Mbit/s, recommended 1GBit/s).
Global Variables	Only global variables from the configuration context may be used!
Permissible Types of Variables	All data types that can be created in SILworX are permitted.
Non-permissible ASCII Characters	The following characters are reserved and must not be used (e.g., for global variables): ! " # ' , . / \` :
HIMax Controllers	An X-OPC server can support a maximum of 255 HIMax controllers.
safeethernet Connection	The X-OPC server can exchange 128 kB in each safe**ethernet** connection.
X OPC Server	10 X-OPC servers can be operated on a host PC.
X-OPC Clients	An X-OPC server supports 10 X-OPC clients.
Data Access Tags	A Data Access server supports a maximum of 100 000 DA tags. Definition: Tags: Data provided by the X-OPC server. Tags correspond to the defined global variables. Items: Data required by the OPC client.
Alarm & Event Event Definitions	An X-OPC Alarm & Event server supports a maximum of 100 000 event definitions.

Table 260: X-OPC Server Properties

10.3 HIMax Controller Properties

Element	Description
safeethernet Connection	The HIMax can exchange a total of 128 kB in each safe**ethernet** connection to the X-OPC server. However, only a view is sent to the X-OPC server in each HIMax cycle. (A view is a fragment of 1100 bytes).
Interfaces	Ethernet 10/100/1000BaseT Processor modules and COM modules Ethernet interfaces in use can simultaneously be used for additional protocols.
Max. number of event definitions	A maximum of 20 000 system events and 6000 E/A events can be defined on a HIMax controller.
Event memory size	A maximum of 5000 events can be stored in the non-volatile event buffer of the HIMax processor module. If the event buffer is full, no new events can be stored as long as at least one X-OPC A&E server has not read an event entry and thus marked it as to be overwritten.
Alarm & Event Timestamp	For each event, the event source can be selected. The events defined as CPU events are created on the processor module. The processor module creates all the events in each of its cycle. This ensures to record and evaluate the value of each global variable as an event. The event defined as I/O events can only be created on SOE I/O modules (e.g., AI 32 02 or DI 32 04). The processor module creates all the events in each of its cycle. Range of values for the UTC timestamp (Universal Time Coordinated): sec fraction since 1970 in [udword] ms fraction of the seconds as [udword] of 0-999 Default value: 2000-01-01 / 00:00:00 An automatic change to/out of daylight-saving time is not supported.
Max. number of X-OPC A&E Server	A maximum of 4 X-OPC A&E servers can access a HIMax controller and simultaneously read events from the event buffer of the processor module.

10.4 Actions Required as a Result of Changes

The following table shows the actions that must be performed after a change in the individual systems.

Type of change	Changes to		
	HIMax	HIMatrix	X-OPC
DA			
Add tags	C+D	C+D	C+D
Tag name (GV change of name)	C+D	C+D	C+D
Delete tags	C+D	C+D	C+D
Change views (parameter and Add/Delete)	C+D	C+D	C+D
A&E			
Add event definition	C+D	n.a.	C+D
Delete Event Definition	C+D	n.a.	C+D
Change Event Source	C+D	n.a.	C+D
Change Alarm Texts	-	n.a.	C+D
Change Alarm Severity	-	n.a.	C+D
Change Ack Required parameter	-	n.a.	C+D
Change Alarm Values with scalar events	C+D	n.a.	C+D
Change the "Alarm at False" parameter with Boolean events	C+D	n.a.	C+D
Change name	C+D	n.a.	C+D
Connect I/O channel with GV	C+R	n.a.	-
Connect state variables to GV	C+R	n.a.	-
In general			
Change safe**ethernet** parameters	C+D	C+D	C+D

Table 261: Actions Required as a Result of Changes

C: Code generation required

R: Reload required

D: Download required

n.a.: non-applicable

-: No action required

10.5 Forcing Global Variables on I/O Modules

i If global variables connected to a process value are forced, the forced global variables have no effect on global variable connected to the parameters
->State LL, L,- N, H, HH.
This particularity also applies if this alarms are configured in the Alarm&Event Editor.
When testing, these variables must be forced individually.

10.6 Configuring an OPC Server Connection

This example shows how to configure a redundant X-OPC server connection to a HIMax controller.

The X-OPC servers provide the process variables and event values of the HIMax controller to the OPC clients. The OPC clients access these process variables and event values and represent them on their user interface.

10.6.1 Software required:

- SILworX
- X OPC Server
- OPC Client

i SILworX is used to configure and operate the entire X OPC server. The X OPC server can be loaded, started and stopped in the SILworX Control Panel like a controller.

10.6.2 Requirements for Operating the X-OPC Server:

- The Ethernet network should have a bandwidth of at least 100 Mbit/s (better 1GBit/s).
- The system time of computer and server must be synchronized, e.g., using SNTP.
- Make sure that the data records for Data Access and Alarm & Events on the controller, X-OPC servers and OPC clients match one another.

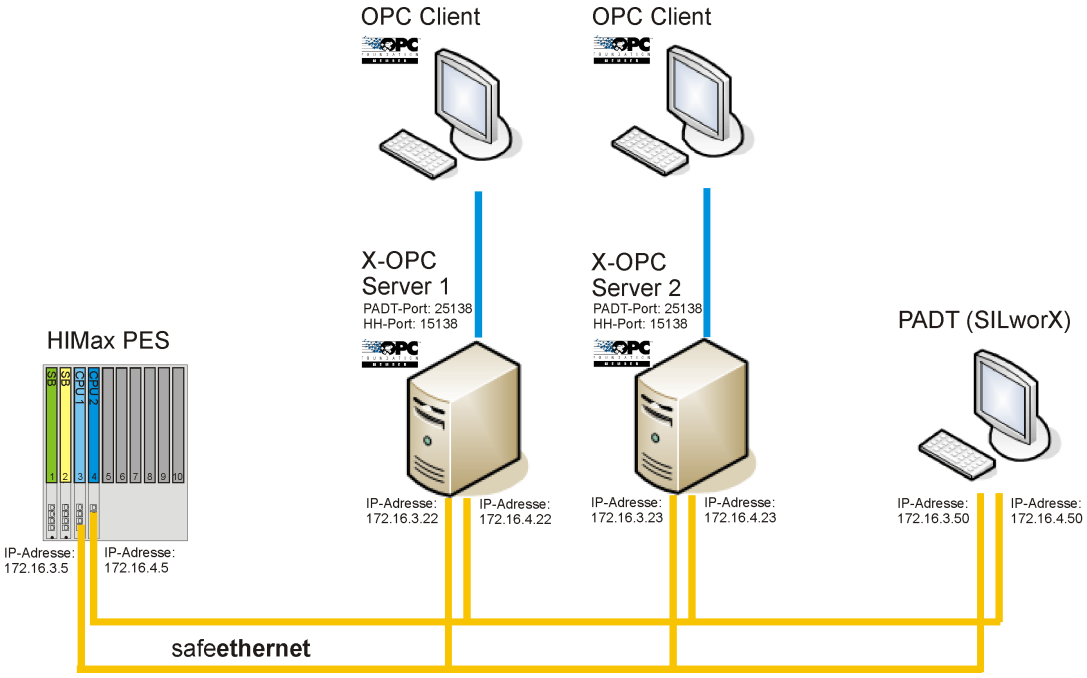


Figure 69: Redundant X-OPC Operation

10.6.3 Installation on Host PC

The X-OPC server must be installed on the respective host PCs.

I Note down the system ID and the number of the PADT port. These values are required for generating the license key!

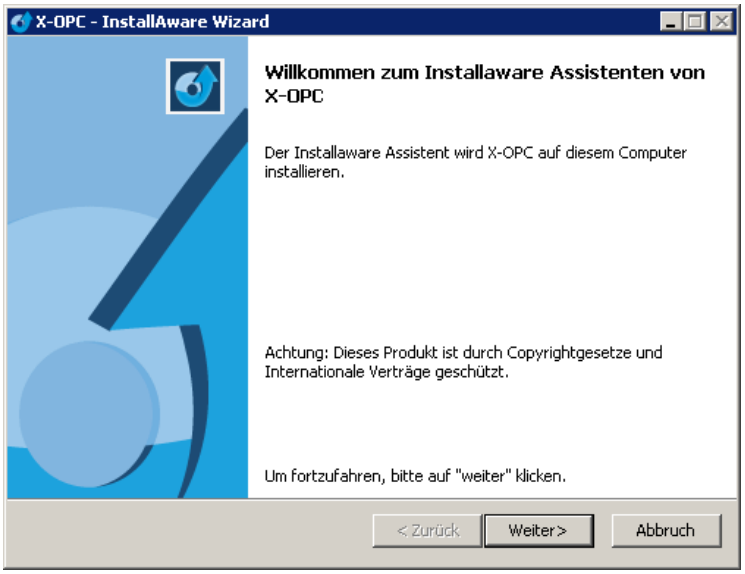


Figure 70: Wizard for Installing the X-OPC Server

To install the X-OPC server on the first host PC

Start the *X-OPC.exe* on each host PC and follow the instructions of the install wizard.

1. Enter the following data for the X-OPC server:
   - System ID: 100
   - PADT port: 25138
   - An arbitrary name for the X-OPC server (it is displayed in the OPC client).
2. To install the X-OPC server, click **Next>**

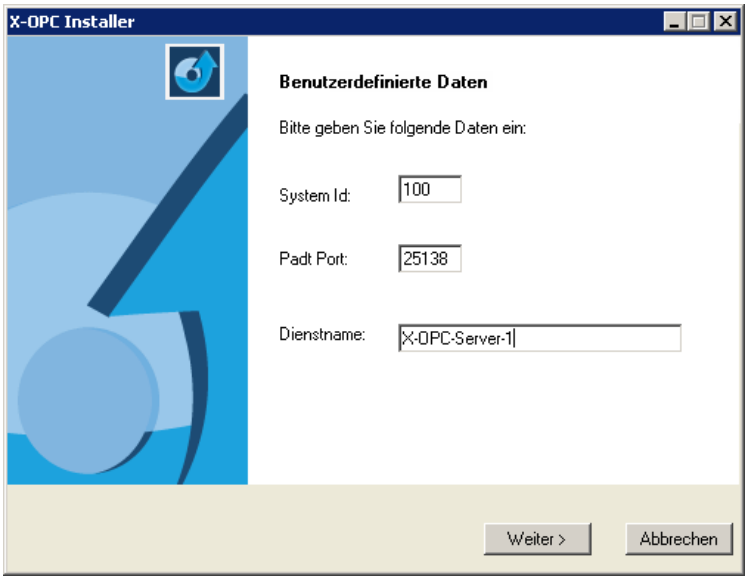


Figure 71: Wizard for Installing the X-OPC Server

To install the X-OPC server on the second host PC

i First determine the class ID of the first X-OPC server prior to installing the second X-OPC server!

If one OPC client is redundantly connected to two X-OPC servers, some OPC client systems expect that the class IDs of the two X-OPC servers are identical. First determine the class ID of the first X-OPC server (e.g., using the OPC client) and note it down.

Start the X-OPC.exe file on the second host PC and follow the instructions of the install wizard.

1. Enter the following data for the X-OPC server:
   - System ID: 110
   - PADT port: 25138
   - An arbitrary name for the X-OPC server (it is displayed in the OPC client).

i PADT port and HH port of the second X-OPC server may be identical with the first one, if the X-OPC servers are run on different PCs.

2. Click **Continue>** to confirm.

To set the same class ID on the second host PC

1. Choose the CLSID setting **manual** for DA and AE.
2. Enter the class ID of the first X-OPC server in the **CLSID** fields.
3. To install the X-OPC server, click **Next>**

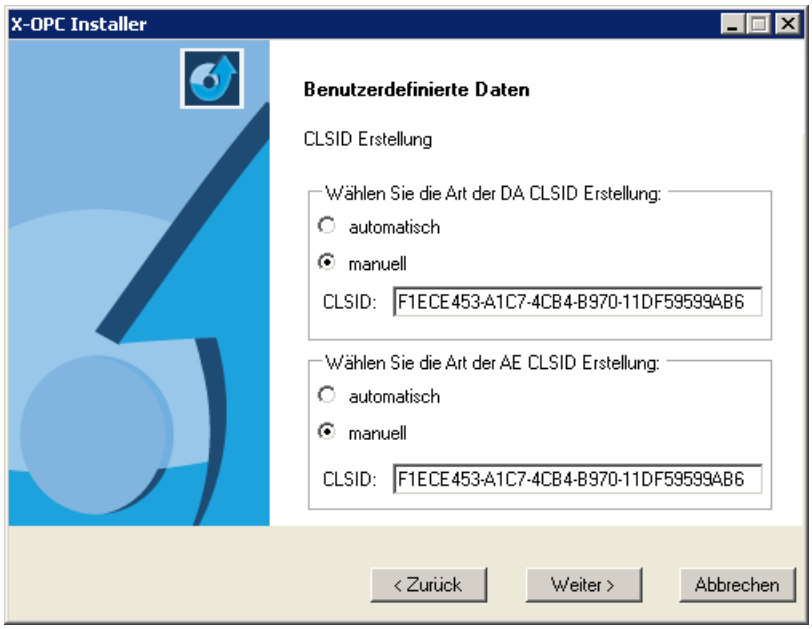


Figure 72: Setting the Class ID of the Second X-OPC Server Manually

To automatically start the X-OPC servers after restarting the PCs

1. In Windows, go to **Start, Settings, Control Panel, Administration, Services** and select **X-OPC Server** from the list.
2. Right-click the OPC Server, then select **Properties**..
3. In the **General** tab, select the **Automatic** start type.

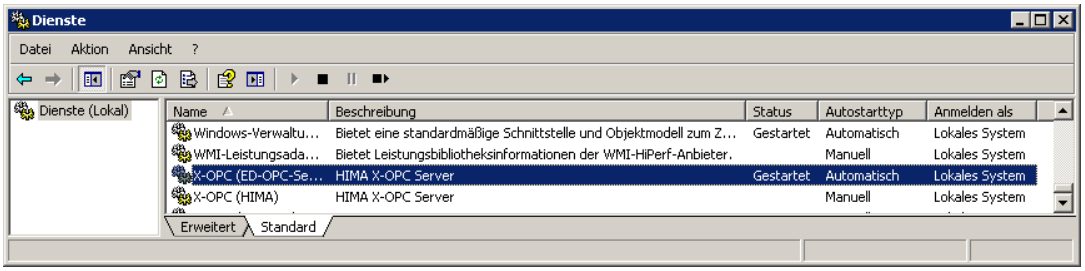


Figure 73: Settings for Starting the X-OPC Server Automatically

To verify if the X-OPC server is running on the PC

1. Open the *Windows Task Manager* and select the *Processes* tab.
2. Check if the *X-OPC.exe* process is running on the PC.

i If the OPC client and OPC server are not running on the same PC, the DCOM interface must be adjusted.

To do this, observe the instructions given in the manual of the OPC Foundation *Using OPC via DCOM witch Microsoft Windows XP Service Pack 2 Version 1.10* (see www.opcfoundation.org).

10.6.4 Configuring the OPC Server in SILworX

To create a new OPC server set in SILworX

1. In the structure tree, open **Configuration.**
2. Right-click **Configuration**, and then click **New, OPC-Server Set**.
   ☑ A new OPC-Server set is added.
3. Right-click OPC Server Set, select **Properties** and accept the default values

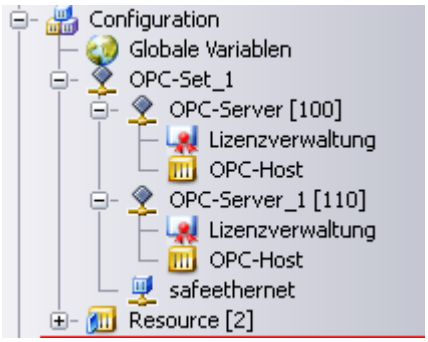


Figure 74: Redundant X-OPC Operation

To configure the first X-OPC server in SILworX

1. In the structure tree, open **Configuration, OPC Server Set**.
2. Right-click **OPC Server Set** and select **New, OPC Server**.
   ☑ A new OPC server is added.
3. Right-click **OPC Server** and select **Properties**.
   - Enter the system ID [SRS] (e.g., 100)
   - Accept the default settings.
3. Right-click **OPC Host** and select **Edit**.
   ☑ The OPC host dialog box for configuring the IP interfaces appears.
4. Right-click anywhere in the OPC host window and select **New IP Device**.
   - Set the PADT port (e.g., 25138).
   - IP address of the PC on which the X-OPC server is installed (e.g., 172.16.3.22).
   - IP address of the PC on which the X-OPC server is installed (e.g., 172.16.4.22).
   - Define it as Standard Interface checking the corresponding checkbox.
   - Set the HH Port (e.g., 15138)

Configure the redundant OPC server in the same OPC server set.

To configure the second OPC server

1. In the structure tree, open **Configuration, OPC Server Set**.
2. Right-click **OPC Server Set** and select **New, OPC Server**.
   ☑ A new OPC server is created.
3. Right-click **OPC Server** and select **Properties**.
   - Enter the system ID [SRS] (e.g., 110)
   - Accept the default settings.
5. Right-click **OPC Host** and select **Edit**.
   ☑ The OPC host dialog box for configuring the IP interfaces appears.
6. Right-click anywhere in the OPC host window and select **New IP Device**.
   - Set the PADT port (e.g., 25138).
   - IP address of the PC on which the X-OPC server is installed (e.g., 172.16.3.23).
   - IP address of the PC on which the X-OPC server is installed (e.g., 172.16.4.23).
   - Define it as Standard Interface checking the corresponding checkbox.
   - Set the HH Port (e.g., 15138)

i If a firewall is installed on the PC, the TCP/UDP PADT and HH ports for the X-OPC servers must be selected on the Exception tab of the firewall configuration.

10.6.5 Settings for the OPC Server in the safeethernet Editor

To create the safeethernet connection between the OPC server and the resource (HIMax controller)

1. In the OPC server set, open the **safeethernet Editor.**
2. In the Object Panel, drag the resource anywhere onto the workspace of the safe**ethernet** Editor.
3. For Alarm & Events, the *Activate SOE* parameter is activated by default.

safeethernet OPC Server-Set_1 *

	Partner	IF CH 1 (lokal)	IF CH 2 (lokal)	IF CH 1 (Ziel)	IF CH 2 (Ziel)	Profil	Sync/Async	R
1	Ressource					Fast & Noisy	ASYNC	
2	Ressource	100.x.x (172.16.3.22:15138)	100.x.x (172.16.3.22:15138)	2.0.5 (172.16.3.5:6010)	2.0.5 (172.16.3.5:6010)			
3	Ressource	110.x.x (122.16.4.23:15138)	110.x.x (122.16.4.23:15138)	2.0.15 (172.16.4.5:6010)	2.0.15 (172.16.4.5:6010)			

Figure 75: Redundant X-OPC Operation

i The used Ethernet interfaces of the PCs are represented in the **IF CH1 (local)** column. The Ethernet interfaces of the HIMax controller must be selected in the **IF CH1 (target)** column.

The safe**ethernet** parameters of the X-OPC server communication are set by default for ensuring the maximum availability.

Receive Timeout = 1000 ms, Response Time = 500 ms etc.

For more information on the safe**ethernet** parameters, refer to Chapter 4.6.

10.6.6 Configuring the X-OPC Data Access Server in SILworX

To create the view definitions for the safeethernet connection

Requirement: The safe**ethernet** Editor of the OPC server must be opened.

1. Right-click the row corresponding to the **resource** to open its context menu.
2. Select **Detail View** to open the detail view of the safe**ethernet** connection.
3. Click the **View Definitions** tab.
4. Right-click anywhere in the workspace and select **New View Definition**.
   The Priority column is used to define how often this view should be sent compared to the other views (a view is a fragment of 1100 bytes).
   For view definitions, first use the standard setting for Priority 1, see also Chapter 10.6.8.
5. Click the **OPC Server Set<->Resource** tab.

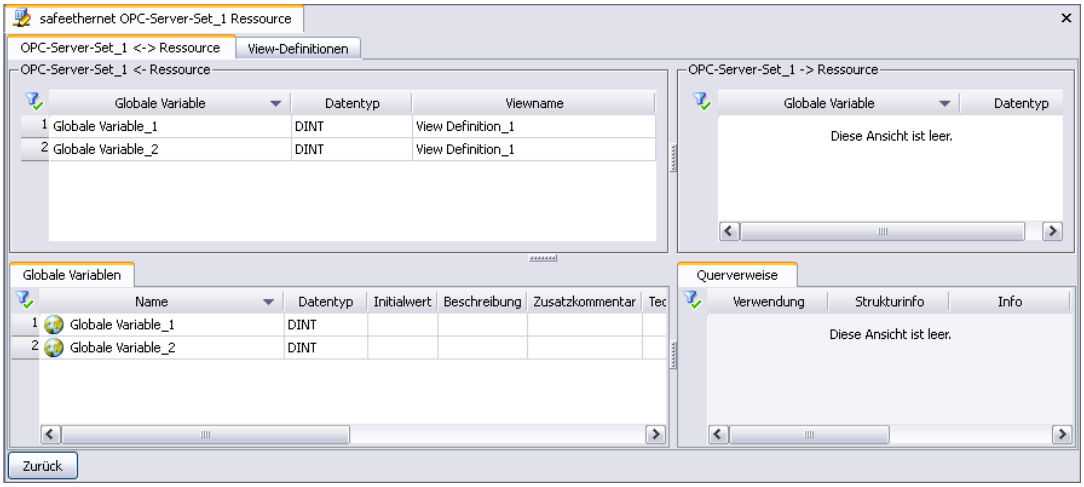


Figure 76: Detailed View of the safe**ethernet** Connection

To add the OPC receive variables

OPC receive variables are sent from the resource to the OPC server.

1. Open the Detail View of the X-OPC safe**ethernet** Editor and select the **OPC Server Set<->Resource** tab.
2. In the Object Panel, drag a **Global Variable** onto the **OPC Server Set <-Resource** area.
3. Double-click the **View Name** column and select the **View Definition** created beforehand..
4. Repeat these steps for every further OPC receive variables.

To add the OPC send variables

OPC send variables are sent from the OPC server to the resource.

1. Open the Detail View of the X-OPC safe**ethernet** Editor and select the **OPC Server Set<->Resource** tab.
2. In the Object Panel, drag a **Global Variable** onto the **OPC Server Set->Resource** area.
3. Double-click the **View Name** column and select the **View Definition** created beforehand..
4. Repeat these steps for every further OPC send variables.

i The OPC send and receive variables must be created in the OPC server set one time only. The variables are automatically used by both X-OPC servers in the OPC server set.

To generate the code and load a resource

1. In the structure tree, select **Configuration, Resource.**
2. Click **Code Generation** in the Action Bar and click **OK** to confirm.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.
4. Load the generated code into the resource..

To generate the code and verify the OPC set

1. In the structure tree, open **Configuration, OPC Server Set.**
2. Click **Code Generation** in the Action Bar and click **OK** to confirm.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

To load the generated code into the X-OPC server

1. Right-click **OPC Server** and select **Online** to perform a **System Login**.
2. Enter the access data:
   - IP address of the PC on which the X-OPC server is installed (e.g., 172.16.3.23).
   - User name: Administrator
   - Password: <empty>
   - Rights: Administrator
3. Click **Login** to open the Control Panel.
4. In the SILworX menu bar, click the **Resource Download** symbol.
   ☑ The code is loaded into the X-OPC server.
5. In the SILworX menu bar, click the **Resource Cold Start** symbol.
   ☑ The X-OPC server is running.

To open the OPC client

The name of the X-OPC server displayed in the OPC client is composed of :
HIMA (manufacturer)**.Service name** (see Chapter 10.6.3) **DA** (Data Access).

Connect to the X-OPC server. At this point, the configured DA data should be transferred to the OPC client.

10.6.7 Configuring the X-OPC Alarm&Event Server in SILworX

This example shows how to connect an X-OPC A&E server to a HIMax controller. The X-OPC A&E server records the events from the HIMax controller via safe**ethernet** and provide them to the OPC client. The OPC client accesses these event variables and displays them on its user interface.

To create an Alarm&Event Editor

1. In the structure tree, open **Configuration, Resource.**
2. Right-click **Resource** and select **New, Alarm&Events**.
   ☑ A new Alarm&Event Editor is created.

To create the Alarm&Events

1. In the structure tree, open **Configuration, Resource.**
2. Right-click **Alarm & Events** and select **Edit**.
3. Select the **Event Definition Bool** tab for Boolean events, see Chapter 10.7.1.
4. Select the **Event Definition Scalar** tab for scalar events, see Chapter 10.7.2.
5. In the Object Panel, drag the **global variable** anywhere in the workspace of the Alarm & Event Editor.
6. Enter the event priority in the safe**ethernet** Editor, see Chapter 10.6.8.

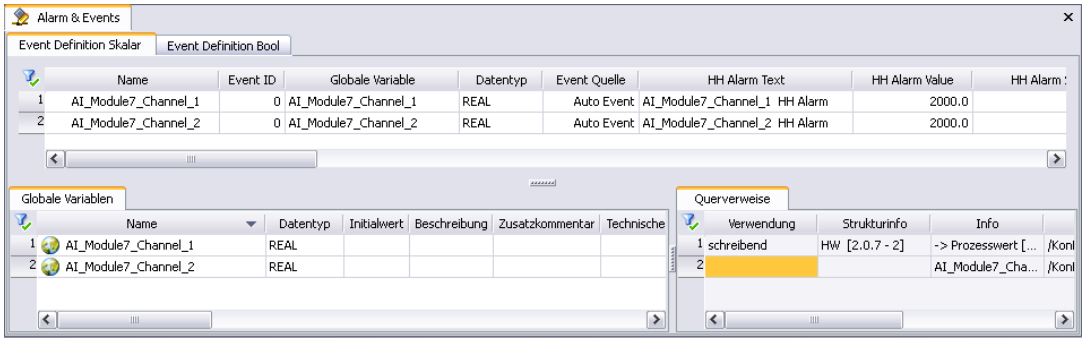


Figure 77: Alarm & Event Editor

To establish the acknowledge connection between both Alarm&Event X-OPC servers:

i If two Alarm&Event X-OPC servers are operated redundantly, the acknowledgements to confirm the alarms on both X-OPC servers can be synchronized. To do this, an acknowledge connection is created.

1. In the structure tree, open **Configuration, OPC Server Set, New.**
2. Right-click **OPC Server Set** and select **New**, **OPC A&E Ack**.
3. Select the following IP connections in the OPC A&E Ack dialog box.
   - IF CH1 (OPC server 1, e.g., 172.16.3.22).
   - IF CH2 (OPC server 1, e.g., 172.16.4.22).
   - IF CH1 (OPC server 2, e.g.,172.16.3.23).
   - IF CH2 (OPC server 2, e.g., 172.16.4.23).

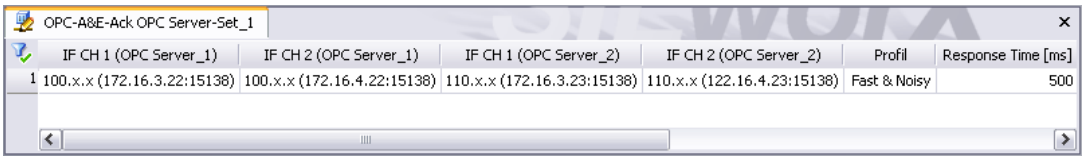


Figure 78: Redundant X-OPC Operation

To generate the code and load a resource

1. In the structure tree, select **Configuration, Resource.**
2. Click **Code Generation** in the Action Bar and click **OK** to confirm.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.
4. Load the generated code into the resource..

To generate the code and verify the OPC set

1. In the structure tree, open **Configuration, OPC Server Set.**
2. Click **Code Generation** in the Action Bar and click **OK** to confirm.
3. Thoroughly verify the messages contained in the Status Viewer and correct potential errors.

To load the generated code into the X-OPC server

1. Right-click the **OPC Server** and select **Online** to log in to the system.
2. Enter the access data:
   - IP address of the PC on which the X-OPC server is installed (e.g., 172.16.3.23).
   - User name: Administrator
   - Password: <empty>
   - Rights: Administrator
3. Click **Login** to open the Control Panel.
4. In the SILworX menu bar, click the **Resource Download** symbol.
   ☑ The code is loaded into the OPC server.
5. In the SILworX menu bar, click the **Resource Cold Start** symbol.
   ☑ The OPC server is running.

To open the OPC client

The name of the X-OPC server displayed in the OPC client is composed of:
HIMA (manufacturer)**.Service name** (see Chapter 10.6.3) **AE (Alarm&Event)**.

Connect to the X-OPC server. At this point, the configured alarms and events should be transferred to the OPC client.

i If a controller and an X-OPC A&E server are connected, the X-OPC A&E server must synchronize with the controller. To do this, the X-OPC A&E server reads the current state of all the variables defined as events and transfers the upcoming alarms to the OPC client. An image of the current controller state can thus be created in the OPC client. The events are only read at this moment.

After the X-OPC server has synchronized with the controller, all the events on the OPC client are updated. Entries for events with an older timestamp are overwritten with the currently read states of the event variables

10.6.8 Configuring the Views and Priorities in SILworX

The HIMax system can send a total of 128 kB for each safe**ethernet** connection to an X-OPC server, but only 1100 bytes per HIMax cycle. To send more data via a safe**ethernet** connection, data must be fragmented. The Priority parameter associated with these fragments (views) can be used to define how often these views should be refreshed.

i Views with **n** priority and views with m priority are sent at a ration of **n:m** times.

For the reaction time from the controller to the X-OPC Server, also observe the number of SOE views and commands (e.g. stopp, start).

$T_R = t_1 + t_2 + t_3 + t_4$ only applies if the priority of all views for state data is equal to 1

- T_R Worst Case Reaction Time
- t_1 Safety Time of PES 1
- t_2 *Number of Views * Receive TMO*
- t_3 Safety Time of X-OPC Server
- t_4 Delay due to SOE function; depending on the number of events and on how the connection is established

The reaction time for the inverse direction can be determined using the same formula, but only one view has usually an effect in this case since the X- OPC server only transfers the data written by OPC clients.

Maximum number of views: 1024

Maximum size of a view: 1100 bytes

Range of values for the priorities: 1 (highest) to 65 535 (lowest)

Designation	Priority default value
Priority of events (Alarm&Event)	1
Priority of state values (Alarm&Event)	10
Priority of the view definitions (Data Access)	1

Table 262: Default Values Associated with the Priorities

Priority of Event Views (Alarm&Event)

The events created by the user in the Alarm&Event Editor are automatically fragmented and transferred in views.

Enter the event priority in the columns **Priority of Events** and **Priority of State Values** of the safe**ethernet** Editor; these priorities are thus valid for all the Alarm&Event views of that given safe**ethernet** connection.

To set the priority values for Alarm&Event Views

1. In the OPC server set, open the **safeethernet Editor**.

Figure 79: safeethernet Editor

2. Double-click Priority of Events to modify the priority of the events.
   All event views are assigned with the priority entered in the **Priority of Events** column (e.g., 1). This is done to define the priority with which the X-OPC server requires events from the controller. If no events exist in the controller at a given time point, none is transferred.
3. Double-click **Priority of State Values** to modify the priority of the event state values. All views for the event state values are assigned with the priority entered in the **Priority of State Values** column (e.g., 10).

i The state values of the events are only required for synchronization reasons (e.g., when the connection is being established); for this reason, they can be transferred in wider intervals than the events.

Data Access View Priorities

The user can assign global variables to each Data Access View and set its priority to determine how often these variables must be refreshed.

To set the priority values for Data Access Views

Requirement: The safe**ethernet** Editor of the OPC server must be opened.

1. Right-click the row corresponding to the **resource** to open its context menu.
2. Select **Detail View** to open the detail view of the safe**ethernet** connection.
3. Click the **View Definitions** tab.
4. Set the priority for Data Access Views in the Priority column.
   - Assign a high priority to Data Access Views with global variables that should be refreshed frequently (e.g., 1).
   - Assign a lower priority to data access views with global variables that should be refreshed less frequently (e.g., 10)

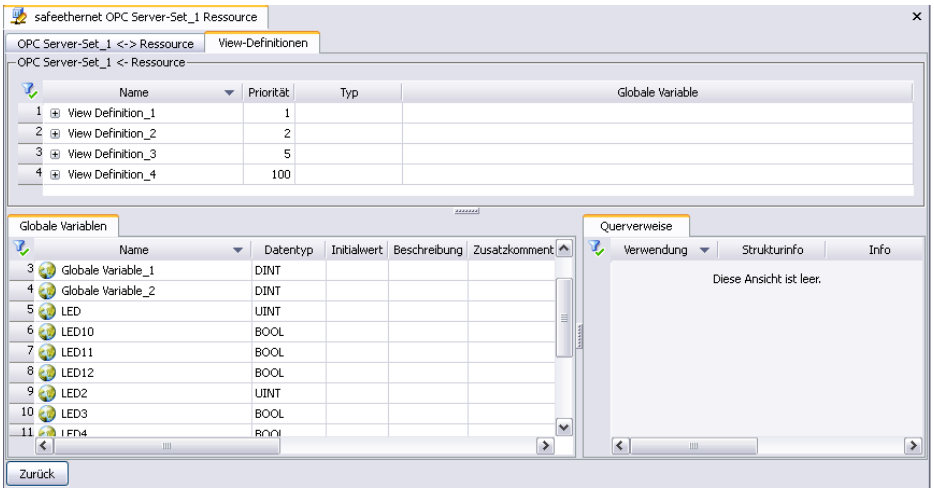


Figure 80: Detailed View of the safe**ethernet** Connection

Further, the state and timestamp of each data access view can be read using the following variables.

Name	Description
View Timestamp [m]	Millisecond fraction of the timestamp (current system time)
View Timestamp [s]	Second fraction of the timestamp (current system time)
View State	See table below

Status	Description
0	CLOSED: Connection is closed
1	TRY OPEN: An attempt is made to open the connection, but it is still closed.
2	CONNECTED: The connection exists and current view data were received (cp. timestamp). As long as no view data is received, the view state is set to TRY_OPEN when establishing the connection.

i The connection state of the safe**ethernet** Editor (see Chapter 4.4) is set to CONNECTED as soon as the connection is open. Unlike View State, no data may have been exchanged here.

Table 263: State and Timestamp for the Data Access Views

10.7 Alarm & Event Editor

The Alarm & Event Editor is used to set the parameters for the alarms and events of the HIMax controller.

To create an Alarm&Event Editor

1. In the structure tree, open **Configuration, Resource.**
2. Right-click **Resource** and select **New, Alarm&Events**.
   ☑ A new Alarm & Events object is added.

To create the Alarm&Events

1. In the structure tree, open **Configuration, Resource.**
2. Right-click **Alarm & Events** and select **Edit**.
3 Select the **Event Definition BOOL** tab for Boolean events, see Chapter 10.7.1.
4. Select the **Event Definition Scalar** tab for scalar events, see Chapter 10.7.2.
5. In the Object Panel, drag the **global variable** anywhere in the workspace of the Alarm & Event Editor.
6. Enter the event priority in the safe**ethernet** Editor, see Chapter 10.6.8.

HIMax differentiate between Boolean and scalar events.

10.7.1 Boolean Events

- Changes of Boolean variables, e.g., of digital inputs.
- Alarm and normal state: They can be arbitrarily assigned to the variable states.

Enter the parameters for the **Boolean** events in the Alarm & Event Editor of the resource; the editor contains the following tabs:

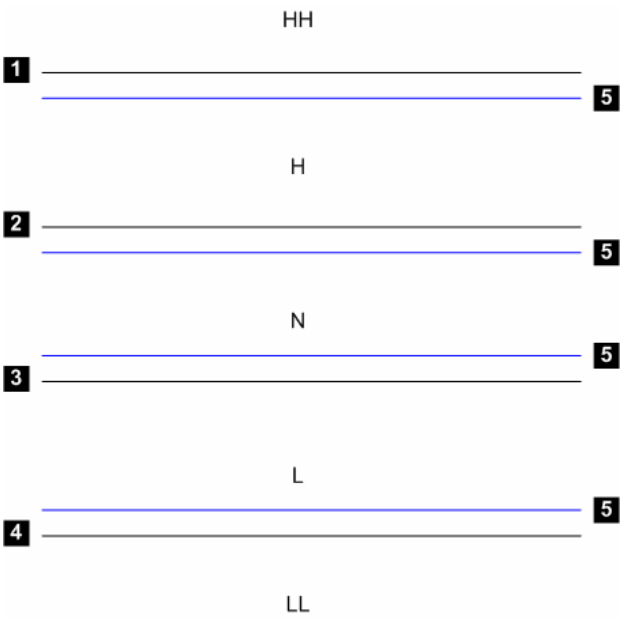
Column	Description	Range of Values
Name	Name of the event definition	Text, max. 31 characters.
Global Variable	Name of the assigned global variable (added using a drag&drop operation)	
Data type	Data type of the global variable; it cannot be modified.	BOOL
Event Source	CPU event The timestamp is created on a processor module. The processor module creates all the events in each of its cycle. IO event Der Zeitstempel wird auf einem geeigneten E/A-Modul gebildet (z. B. DI 32 04). Auto event A CPU event and, if available, IO events of the I/O module are created. Default value: CPU event	CPU Event, IO Event, Auto Event
Alarm when FALSE	Activated If the global variable value changes from TRUE to FALSE, an event is triggered. Deactivated If the global variable value changes from FALSE to TRUE, an event is triggered. Default value: Deactivated	Checkbox activated, deactivated
Alarm Text	Text specifying the alarm state	Text
Alarm Severity	Priority of the alarm state Default value: 1	1...1000

Alarm Ack Required	Activated The alarm state must be confirmed by the user (acknowledgement) Deactivated The alarm state may not be confirmed by the user Default value: Deactivated	Checkbox activated, deactivated
Return to Normal Text	Text specifying the alarm state	Text
Return to Normal Severity	Priority of the normal state	1...1000
Return to Normal Ack Required	The normal state must be confirmed by the user (acknowledgement) Default value: Deactivated	Checkbox activated, deactivated

Table 264: Parameters for Boolean Events

10.7.2 Scalar Events

- Exceedance of the limits defined for a scalar variable, e.g., an analog input.
- Two upper limits and two lower limits are possible.
  For the limit values, the following condition must be met:
  Superior limit ≥ upper limit ≥ normal area ≥ lower limit ≥ inferior limit.
  An hysteresis is effective in the following cases:
  - If the value falls below the upper limit
  - If the value exceeds the lower limit
  An hysteresis is defined to avoid a needless large number of events when a global variable strongly oscillate around a limit.



1 HH Alarm Value
2 H Alarm Value
3 L Alarm Value
4 LL Alarm Value
5 Hysteresis

Figure 81: Five Areas of a Scalar Event

Enter the parameters for the scalar events in the Alarm & Event Editor of the resource; the editor contains the following tabs:

Column	Description	Range of Values
Name	Name of the event definition	Text, max. 31 characters.
Global Variable	Name of the assigned global variable (added using a drag&drop operation)	
Data type	Data type of the global variable; it cannot be modified.	depending on the global variable type
Event Source	CPU event The timestamp is created on a processor module. The processor module creates all the events in each of its cycle. IO event The timestamp is built on an appropriate I/O module (e.g., AI 32 02). Auto event A CPU event and, if available, IO events of the I/O module are created. Default value: CPU Event	CPU Event, IO Event, Auto Event
HH Alarm Text	Text specifying the alarm state of the upper limit.	Text
HH Alarm Value	Upper limit triggering an event. Condition: (HH Alarm Value - Hysteresis) > H Alarm Value or HH Alarm Value = H Alarm Value	depending on the global variable type
HH Alarm Severity	Priority of the upper limit; default value: 1	1...1000
HH Ack Required	Activated The user must confirm that the upper limit has been exceeded (acknowledgment). Deactivated The user may not confirm that the upper limit has been exceeded. Default value: Deactivated	Checkbox activated, deactivated
H Alarm Text	Text specifying the alarm state of the upper limit.	Text
H Alarm Value	Upper limit triggering an event. Condition: (H Alarm Value - Hysteresis) > (L Alarm Value + Hysteresis) or H Alarm Value = L Alarm Value	depending on the global variable type
H Alarm Severity	Priority of the upper limit; default value: 1	1...1000
H Ack Required	Activated The user must confirm that the upper limit has been exceeded (acknowledgment). Deactivated The user may not confirm that the upper limit has been exceeded. Default value: Deactivated	Checkbox activated, deactivated
Return to Normal Text	Text specifying the alarm state	Text
Return to Normal Severity	Priority of the normal state; default value: 1	1...1000
Return to Normal Ack Required	The normal state must be confirmed by the user (acknowledgement); default value: Deactivated	Checkbox activated, deactivated
L Alarm Text	Text specifying the alarm state of the lower limit.	Text
L Alarm Value	Lower limit triggering an event. Condition: (L Alarm Value + Hysteresis) < (H Alarm Value - Hysteresis) or L Alarm Value = H Alarm Value	depending on the global variable type

L Alarm Severity	Priority of the lower limit; default value: 1	1...1000
L Ack Required	Activated The user must confirm that the lower limit has been exceeded (acknowledgment). Deactivated The user may not confirm that the lower limit has been exceeded. Default value: Deactivated	Checkbox activated, deactivated
LL Alarm Text	Text specifying the alarm state of the lowest limit.	Text
LL Alarm Value	Lower limit triggering an event. Condition: (LL Alarm Value + Hysteresis) < (L Alarm Value) or LL Alarm Value = L Alarm Value	depending on the global variable type
LL Alarm Severity	Priority of the lowest limit; default value: 1	1...1000
LL Ack Required	Activated The user must confirm that the lowest limit has been exceeded (acknowledgment). Deactivated The user may not confirm that the lowest limit has been exceeded. Default value: Deactivated	Checkbox activated, deactivated
Alarm Hysteresis	The hysteresis avoids that many events are continuously created when the process value often oscillate around a limit.	depending on the global variable type

Table 265: Parameters for Scalar Events

10.8 Parameters for the X-OPC Server Properties

i SILworX is used to configure and operate the entire X OPC server. The X OPC server can be loaded, started and stopped in the SILworX Control Panel like a controller.

10.8.1 OPC Server Set

The OPC set is used as common platform for configuring up to two OPC servers.

The OPC Server Set properties for the two redundant X-OPC servers are automatically set identical.

To create a new OPC server set

1. In the structure tree, open **Configuration**.
2. Right-click **Configuration** and select **New, OPC Server Set** to create a new OPC server set.
3. Right-click **OPC Server Set** and select **Properties**. Accept the default values.

The Properties dialog box for the OPC server set contains the following parameters:

Element	Description
Name	Name of the OPC server set. A maximum of 31 characters.
Safety Time [ms]	The safety time is the time in milliseconds within which the X-OPC server must react to an error. Condition: safety time ≥ 2 * watchdog time Range of values: 2000...400 000 ms Default value: 20 000 ms
Watchdog Time [ms]	The watchdog time is the maximum time that the OPC server may require to complete a program cycle. If the defined watchdog time is exceeded (the execution of a program cycle takes too long), the X-OPC server is stopped. Condition: WDT ≥ 1000 ms and ≤ 0,5 * safety time Range of values: 1000...200 000 ms Default value: 10 000 ms

Main Enable	The setting of the 'Main Enable' OPC switch affects the function of the other OPC switches. If 'Main Enable' is deactivated, the parameters set for the other OPC switches cannot be modified while the user program is being processed (the controller is in RUN). Default value: Activated
Autostart	Autostart defines if the OPC configurations may be automatically started with a cold start or a warm start after powering up or booting the controller or should not be started (Off). If Autostart is deactivated, the X-OPC server adopts the STOP/VALID CONFIGURATION state after booting. Default value: Deactivated
Start Allowed	Only if *Start Allowed* is activated, an X-OPC server can be started from within the programming tool. If *Start Allowed* is deactivated, the X-OPC server cannot be started from within the programming tool. In such a case, the X-OPC server can only be started if *Autostart* is activated and the host PC is started or rebooted. If neither *Autostart* nor *Start Allowed* is activated, the X-OPC server cannot be started. This can be required, for instance, during maintenance actions to prevent the system from starting. Default value: Activated
Load Allowed	If *Load Allowed* is deactivated, no (new) OPC configuration can be loaded into the controller. Deactivate *Loading Allowed* to avoid that the OPC configuration loaded into the X-OPC server is overwritten. Default value: Activated
Reload Allowed	No function yet!
Global Forcing Allowed	*Global forcing* can only be started if *Global Forcing Allowed* is activated. i The Force Editor can also be used to display variable contents if *Global Forcing Allowed* is deactivated. Default value: Deactivated

Global Force Timeout Reaction	If *Global Force Timeout Reaction, Stop Resource* is selected, the X-OPC server enters the STOP state after the preset force time has expired. All the outputs of the X-OPC server are set to LOW. If *Global Force Timeout Reaction, Stop Forcing Only* is selected, the X-OPC server continues executing the OPC configuration after the force time has expired. i If forcing is allowed, carefully check the setting for 'Stop at Force Timeout'. Also observe the instructions provided in the Safety Manual. Default value: Stop Resource
Max.Com. Time Slice ASYNC [ms]	*Max. Com. Time Slice ASYNC [ms]* is the time in milliseconds that is reserved in each X-OPC server cycle for performing all the communication tasks scheduled for P2P communication. Default value: 500 ms
Target Cycle Time [ms]	Target cycle time for the X-OPC Server Default value: 50 ms
safeethernet CRC	
Namespace Separator	Dot . Slash / Colon : Backslash \ Default value: Dot
Namespace Type	Depending on the OPC client requirements, the following name space types can be set: ▪ Hierarchical Namespace ▪ Flat Namespace Default value: Hierarchical name space
Short tag names for DA	This parameter can only be activated if *Flat Namespace* is selected. It is an option in which data and event are offered to the OPC client without any further context (path name). Default value: Deactivated
Changeless update	Setting according to the OPC client requirement Activated: If *Changeless Update* is selected and 'OPC Group UpdateRate' has expired, the X-OPC server provides all items to the OPC client. Deactivated: If *Changeless Update* is not selected,only the modified values are provided to the OPC client (this behavior is in accordance with the OPC Specification.

Cycle Delay [ms]	The cycle delay limits the CPU load of the PC due to the X-OPC server to allow other programs to be run. Range of values: 1...100 ms Default value: 5 ms
Simple-Events for CPU I/O Events	Never Only at Start Allways
Short Tag Names for I&O	This parameter can only be activated if *Flat Namespace* is selected. It is an option in which data and event sources are offered to the OPC client without any further context (path name). Default value: Deactivated

Table 266: Properties

10.8.2 OPC Server

To create a new OPC server

1. In the structure tree, open **Configuration, OPC Server Set**.
2. Right-click **OPC Server Set** and select **New, OPC Server** to create a new OPC server.
3. Right-click **OPC Server** and select **Properties**.

The Properties dialog box for the OPC server contains the following parameters:

Element	Description
Name	Name for the OPC server. A maximum of 31 characters.
System ID [SRS]	Default value: 60000

Table 267: Properties

To open the OPC host

1. In the structure tree, open **Configuration, OPC Server Set, OPC Server**.
2. Right-click **OPC Host** and select **Edit** to get an overview of the IP interfaces.

The Edit dialog box for the OPC host contains the following parameters:

Element	Description
PADT Port	Default value: 25138
Name	Name for the OPC server set. A maximum of 31 characters.
IP Address	IP address of the host PC. Default value: 192.168.0.1
Standard Interface	It must be selected if the host PC is equipped with more than one Ethernet port. Default value: Activated
HH Port	Default value: 15138

Table 268: Edit

10.9 Uninstalling the X-OPC Server

To uninstall the X-OPC server

1. Open **Start, Settings, Control Panel, Software** in Windows.
2. From the list, select the X-OPC server to be uninstalled and click the **Remove** button.
3. Follow the instructions of the Uninstall Wizard.

11 ComUserTask

In addition to the user program created in SILworX, a C program can also be run in the controller.

With no interaction with the safe processor module, this non-safe C program runs as a ComUserTask on the communication module of the controller.

The ComUserTask has its own cycle time that does not depend on the CPU cycle.

This allows the users to program in C any kind of applications and implement them as ComUserTask, for instance:

- Communication interfaces for special protocols (TCP, UDP, etc.).
- Gateway function between TCP/UDP and serial communication.

11.1 System Requirements

Equipment and system requirements

Element	Description
Controller	HIMax with COM module
Processor module	The Ethernet interfaces on the processor module may not be used for ComUserTask.
COM module	Ethernet 10/100BaseT Pin assignment of the D-sub connectors FB1 and FB2 e.g., for RS 232. If the serial fieldbus interface (FB1 or FB2) are used, they must be equipped with an optional HIMA submodule, see Chapter 3.4.
Activation	Software activation code required, see Chapter 3.5.

Table 269: Equipment and System Requirements for the ComUserTask

ComUserTask Properties

Element	Description
ComUserTask	One ComUserTask can be configured for each HIMax controller.
Safety-related	No
Data Exchange	Configurable
Code and data area	Start address 0x790000 Length 448 kBytes
Stack	The stack is located in a reserved memory outside the code/data area. Length 64 kByte

Table 270: ComUserTask Properties

11.1.1 Creating a ComUserTask

To create a new ComUserTask

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, click **New, ComUserTask** to add a new ComUserTask.
3. Right-click the ComUserTask, click **Properties** and select the **COM Module**. Accept the default settings for the first configuration.

11.2 Requirements

In addition to the normal C commands, a specific library with defined functions is available for programming a ComUserTask (see Chapter 11.4).

Development Environment

The development environment comprises the GNU C Compiler and Cygwin which are available on a separate installation CD (not included in ELOP II Factory) and are subject to the conditions of the GNU General Public License (see www.gnu.org).

The newest versions and documents for the development environment can be downloaded from the corresponding Internet pages at www.cygwin.com and www.gnu.org.

Controller

In the HIMax/ HIMatrix controllers the ComUserTask has no access to the safe hardware inputs and outputs. If an access to the safe hardware inputs and outputs is required, a CPU user program must exist for connecting the variables (see 11.4.5).

11.3 Abbreviations

Abbreviation	Description
CUCB	COM User Callback (CUCB_ Functions invoked by the COM)
CUIT	COM User IRQ Task
CUL	COM User Library (CUL_ Functions invoked in the CUT)
CUT	ComUserTask
GNU	GNU project
IF	Interface
FB	Field bus interface of the controller
FIFO	First In First Out (Data memory)
NVRam	Non volatile random access memory, non volatile memory

Table 271: Abbreviations

11.4 CUT Interface in SILworX

The process data communication of the ComUserTask runs between COM and CPU.

⚠ WARNING

The loaded ComUserTask must not use privileged commands of the COM module.

The code of the ComUserTask runs in the COM in a non-reactive way to the safe CPU. This ensures that the safe CPU is protected against the CUT code. Note that errors in the CUT code can disturb the entire COM function, thus affecting or stopping the controller's function. The CPU safety functions, however, are not compromised.

11.4.1 Schedule Interval [ms]

The ComUserTask is invoked in a predefined schedule interval [ms] during the controller's states RUN and STOP_VALID_CONFIG (COM module).

In SILworX, Schedule Interval [ms] can be set in the ComUserTask *Properties*.

Schedule Interval [ms]	
Range of values:	10 .. 255 ms
Default value:	15 ms

Table 272: Schedule Interval [ms]

i The COM processor time available to the CUT depends on the other configured COM functions such as safe**ethernet** or Modbus TCP.

If CUT is not finished within the schedule interval, each call to restart the CUT is ignored until CUT has been processed.

11.4.2 Scheduling Preprocessing

In the controller's state RUN:

Before each CUT call, the COM module provides the process data of the safe CPU to the CUT in a memory area defined by CUT.

In the controller's state STOP:

No process data is exchanged between COM and safe CPU.

11.4.3 Scheduling Postprocessing

In the controller's state RUN:

After each CUT call, the COM module provides the process data of the CUT to the safe CPU.

In the controller's state STOP:

No process data is exchanged between COM and safe CPU.

11.4.4 STOP_INVALID_CONFIG

If the COM is in the STOP_INVALID_CONFIG state, CUT is not executed.

If the COM module enters the STOP_INVALID_CONFIG state and executes CUT or CUIT, these functions are terminated.

11.4.5 CUT Interface Variables (CPU<->CUT)

Configuring not safety-related process data communication between safe CPU and COM (CUT).

Send Direction	Max. Process Data Size
COM->CPU	16375 bytes data (16384 bytes – 9 status bytes)
CPU->COM	16382 bytes data (16384 bytes – 2 control bytes)

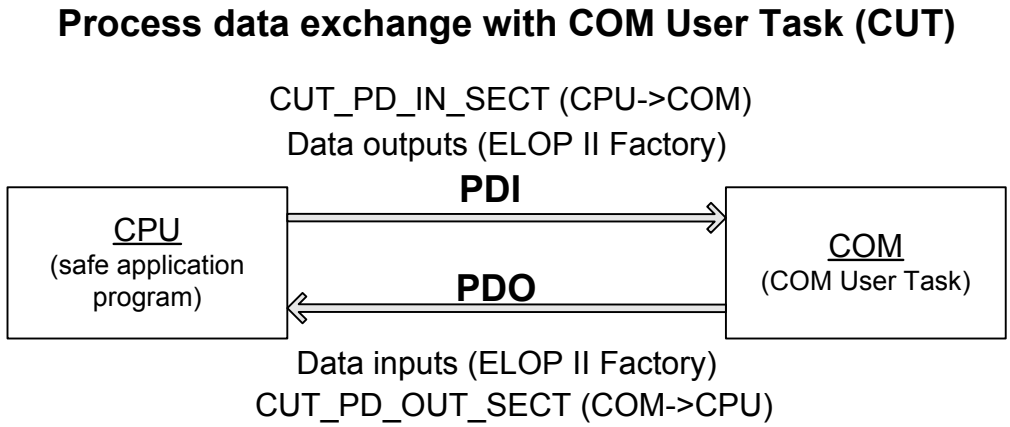


Figure 82: Process Data Exchange between CPU and COM (CUT)

All data types that are used in SILworX can be exchanged.

The data structure must be configured in SILworX.
The size of the data structures CUT_PDI and CUT_PDO (in the compiled CUT C code) must correspond to the size of the data structure configured in SILworX.

i If the data structures CUT_PDI and CUT_PDO are not available in the compiled C code or do not have the same size as the data structure of the process data configured in SILworX, the configuration is invalid and the COM module enters the STOP_INVALID_CONFIG state.

Process data communication takes place in the RUN state only.

The **Edit** menu function opens the tabs Process Variables and System Variables.

System Variables

The **System Variables** tab contains the following system parameters for monitoring and controlling the CUT:

<table>
<tr><th>Name</th><th>Function</th></tr>
<tr><td>Execution Time
[DWORD]</td><td>Execution time of the ComUserTask in µs</td></tr>
<tr><td>Real schedule interval
[DWORD]</td><td>Time between two ComUserTask cycles in ms</td></tr>
<tr><td>User task state control
[WORD]</td><td>The following table shows how the user can control the ComUserTask with the User Task State Control system parameter:
<table>
<tr><th>Function</th><th>Description</th></tr>
<tr><td>DISABLED
0x8000</td><td>The application program locks the CUT (CUT is not started).</td></tr>
<tr><td>Autostart
Default: 0</td><td>After termination a new start of the CUT is automatically allowed if the error is eliminated.</td></tr>
<tr><td>TOGGLE_MODE_0
0x0100</td><td>After termination of the CUT a new start of the CUT is only allowed after writing TOGGLE_MODE_1.</td></tr>
<tr><td>TOGGLE_MODE_1
0x0101</td><td>After termination of the CUT a new start of the CUT is only allowed after writing TOGGLE_MODE_0.</td></tr>
</table></td></tr>
<tr><td>State of the User Task
[BYTE]</td><td>1 = RUNNING (CUT is running)
0 = ERROR (CUT is not running due to an error)</td></tr>
</table>

Table 273: ComUserTask System Variables

Process Variables

Input Signals (COM->CPU)

The **Input Signals** tab contains the signals that should be transferred from the COM module (CUT) to the CPU (CPU input area).

⚠ CAUTION

Non-safe data of ComUserTask!

Non-safe variables of the ComUserTask must not hinder the safety functions of the CPU user program.

Input signals of ComUserTask

Name	Data type	Offset
CUT_Counter	DWORD	0
Time_Stamp	DWORD	4

Table 274: Input signals of ComUserTask

Required entry in the C code

The C code of the COM User Tasks must have the following CUT_PDO data structure for the COM outputs (CPU input area):

```
/* SILworX Input Records (COM->CPU) */
uword CUT_PDO[1] __attribute__ ((section("CUT_PD_OUT_SECT"), aligned(1)));
```

The size of the CUT_PDO data structure must correspond to the size of the data inputs configured in SILworX.

Output Signals (CPU->COM)

The **Output Signals** tab contains the signals that should be transferred from the CPU (CPU output area) to the COM module (CUT).

ComUserTask Output Signals

Name	Data type	Offset
CPU/COM_1	WORD	0
CPU/COM_2	WORD	2

Table 275: ComUserTask Output Signals

Required entry in the C code

The C code of the ComUserTask must have the following CUT_PDI data structure for the COM inputs (CPU output area):

```
/* SILworX Output Records (CPU->COM) */
uword CUT_PDI[1] __attribute__ ((section("CUT_PD_IN_SECT"), aligned(1)));
```

The size of the CUT_PDI data structure must correspond to the size of the data outputs configured in SILworX.

11.5 CUT Functions

11.5.1 COM User Callback Functions

The COM User callback functions have all the **CUCB _** prefix and are directly invoked from the COM when events occur.

i All COM user callback functions must be defined in the user's C code!
Also the CUCB_IrqService function, which is not supported for HIMax and HIMatrix, must be defined in the user C-Code for reasons of compatibility.
Funktion prototype: void CUCB_IrqService(udword devNo) {}

The COM user callback (CUCB) and the COM user library (CUL) functions share the stack and the same code and data memory. These functions mutually ensure the consistency of the data shared (variables).

11.5.2 COM User Library Functions

All COM user library functions and variables have the **CUL_** prefix and are invoked in the CUT.

All the CUL functions are available via the **libcut.a** object file.

11.5.3 Header Files

The two header files **cut.h** and **cut_types.h** contain all function prototypes for CUL/CUCB and the related data types and constants.

For the data types, the following short cuts are defined in the header file **cut_types.h**:

```
typedef unsigned long     udword;
typedef unsigned short    uword;
typedef unsigned char     ubyte;
typedef signed long       dword;
typedef signed short      word;
typedef signed char       sbyte;
#ifndef HAS_BOOL
typedef unsigned char     bool; // with 0=FALSE, otherwise TRUE
#endif
```

11.5.4 Code/Data Area and Stack for CUT

The code/data area is a coherent memory area that begins with the code segment and the initial data segment and continues with the data segments. In the HIMA linker files (**makeinc.inc.app** and **section.dld**), the written segment sequence and the available storage capacity are predefined.

The ComUserTask uses the HIMA linker files to optimally distribute the available area among code and data.

Start Address 0x790000

Length 448 kByte

The stack is located in a reserved memory area defined when the COM operating system is started.

End address Dynamically (from the point of view of CUT)

Length 64 kByte

11.5.5 Start Function CUCB_TaskLoop

`CUCB_TaskLoop()` is the starting function associated with the ComUserTask.

The ComUserTask program execution begins when this function is invoked (see Chapter 11.4.1 `Schedule Interval[ms]`).

Function Prototype:

void CUCB_TaskLoop(udword mode)

Parameter:

The function has the following parameter:

Parameter	Description
mode	1 = MODE_STOP corresponds to the mode STOP_VALID_CONFIG 2 = MODE_RUN controller's normal operation

Table 276: Parameter

11.5.6 RS485 / RS232 IF Serial Interfaces

The used fieldbus interfaces must be equipped with the corresponding fieldbus submodules (hardware).

---

i For each HIMax controller, the user can refer to the corresponding system documentation.

CUL_AscOpen

The `CUL_AscOpen()` function initializes the entered serial interface (*comId*) with the given parameters. After invoking the `CUL_AscOpen()` function, the COM immediately begins receiving data via this interface.

The received data is stored in a FIFO software with a size of 1kByte for **each** initialized serial interface.

The data is stored until it is read out with the CUT function `CUL_AscRcv()`.

---

i If data is read out of the FIFO more slowly than it is received, the new data is rejected.

Function Prototype:

```
udword CUL_AscOpen(  Udword comId,
                     Ubyte duplex,
                     udword baudRate,
                     ubyte parity,
                     ubyte stopBits)
```

Parameter:

The function has the following parameters:

Parameter	Description
comId	Field bus interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE
duplex	0 = Full duplex (only permitted for FB4 if RS232) 1 = Half duplex
baudRate	1 = 1200 Bit 2 = 2400 Bit 3 = 4800 Bit 4 = 9600 Bit 5 = 19200 Bi 6 = 38400 Bi 7 = 57600 Bi 8 = 115000 Bit
The data bit length is fixed and set to 8 data bits. Start, parity and stop bits must be added to these 8 data bits.	
parity	0 = NONE 1 = EVEN 2 = ODD
stopBits	1 = 1 Bit 2 = 2 Bits

Table 277: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
CUL_OKAY	The interface was initialized successfully.
CUL_ALREADY_IN_USE	The interface is used by other COM functions or is already open.
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors

Table 278: Return Value

CUL_AscClose

The `CUL_AscClose()`function closes the serial interface entered in *comId*. In doing so, the data that has already been received but not read out with the function CUL_AscRcv() is deleted in FIFO.

Function Prototype:

Udword CUL_AscClose(udword comId)

Parameter:

The function has the following parameter:

Parameter	Description
comId	Field bus interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE

Table 279: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
CUL_OKAY	The interface was closed successfully .
CUL_NOT_OPENED	The interface was not opened (by the CUT).
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors

Table 280: Return Value

CUL_AscRcv

The `CUL_AscRcv( )`function instructs the COM to provide a defined data volume from the FIFO.

As soon as the requested data is available (and the CUL or the scheduling allows it), the COM invokes the *CUCB_AscRcvReady()* function .

If not enough data is contained in FIFO, the `CUL_AscRcv( )`function returns immediately.

The instruction to receive data is stored until:

- The instruction was completely processed or
- the *CUL_AscClose()* function is invoked or
- redefined due to a new instruction

i Until the instruction is completely processed, the *pBuf content may only be changed using the *CUCB_AscRcvReady()* function.

Function Prototype:

Udword CUL_AscRcv(udword comId, CUCB_ASC_BUFFER *pBuf)

```
typedef struct CUCB_AscBuffer {

      bool  bAscState;  // for using by CUT/CUCB
      bool  bError;     // for using by CUT/CUCB
      uword align;      // COM is 4 aligned, long's are higher-performance
      udword mDataIdx;  // Byte offset in aData from there on the data are loacte
      udword mDataMax;  // max. Byte offset: (mDataMax-mDataIdx) indicates,
                        // how many bytes in aData have to be sent or received
                        // Start point of the data copy range
      udword aData[1];
      }CUCB_ASC_BUFFER;
```

Parameter:

The function has the following parameters:

Parameter	Description
comId	Field bus interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE
pBuf	It defines the requested amount of data and the location, to which the data should be copied, before CUCB_AscReady() is invoked. If sufficient data has already been received in the FIFO, the CUCB_AscRcvReady() function is invoked during CUL_AscRcv().

Table 281: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
CUL_OKAY	If the order was successful, otherwise error code.
CUL_NOT_OPENED	If the interface was not opened by the CUT.
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors

Table 282: Return Value

Restrictions:

- If the memory area (defined by CUCB_ASC_BUFFER) is not allocated in the CUT data segment, CUT and CUIT are terminated.
- A maximum of 1024 bytes of data can be requested.

CUCB_AscRcvReady

If the COM module invokes calls the `CUCB_AscRcvReady()`function, the requested amount of data is ready in FIFO (data from the serial interface defined in the `comId` parameter).

The data has been previously requested with the `CUL_AscRcv()`function.

The `CUCB_AscRcvReady()`function can be invoked before, after or while invoking the `CUL_AscRcv()`function. The task context is always that of CUT.

The `CUCB_AscRcvReady()`function may invoke all CUT library functions.

These actions are also permitted:

- the increase of `mDataMax`, and/or
- Reconfiguring `mDataIdx` and `mDataMax` by the `*.pBuf` data assigned to `comId` (for further reading)

The structure element of CUCB_ASC_BUFFER.mDataIdx has the value of CUCB_ASC_BUFFER.mDataMax.

Function Prototype:

void CUCB_AscRcvReady(udword comId)

Parameter:

The function has the following parameter:

Parameter	Description
comId	Field bus interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE

Table 283: Parameter

Restrictions:

If the memory area (defined by CUCB_ASC_BUFFER) is not located in the CUT data segment, CUIT and CUT are terminated.

CUL_AscSend

The CUCB_AscSend function sends the data set defined by the parameter pBuf via the serial interface comId.

The defined data set must be ≥ 1 byte and ≤ 1kByte.

After data have been sent, the CUCB_AscSendReady( )function is invoked. If an error occurs,

- it is not be sent and
- the CUCB_AscSendReady( )function will not be invoked.

Function Prototype:

Udword CUL_AscSend(udword comId, CUCB_ASC_BUFFER *pBuf)

Parameter:

The function has the following parameters:

Parameter	Description
comId	Field bus interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE
pBuf	Defines the data amount to be sent

Table 284: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
CUL_OKAY	If data sending was successfully
CUL_WOULDBLOCK	If a message previously sent has not been sent yet
CUL_NOT_OPENED	If the interface was not opened by the CUT
CUL_INVALID_PARAM	Incorrect parameters or parameter combinations transmitted.
CUL_DEVICE_ERROR	Other errors

Table 285: Return Value

Restrictions:

If the memory area (defined by CUCB_ASC_BUFFER) is not located in the CUT data segment, CUIT and CUT are terminated.

CUCB_AscSendReady

If the COM invokes the `CUCB_AscSendReady()` function, data is completely sent with the `CUCB_AscSend()` function via the serial interface.

The task context is always that of CUT. The `CUCB_AscSendReady()` function may invoke all CUT library functions.

Function Prototype:

void CUCB_AscSendReady(udword comId)

Parameter:

The function has the following parameter:

Parameter	Description
comId	Field bus interface (RS485, RS 232) 1 = FB1, 2 = FB2, 3 = FB3, 4 = FB4_SERVICE

Table 286: Parameter

11.5.7 UDP/TCP Socket IF

A maximum of 8 sockets can simultaneously be used irrespective of the used protocol.

The physical connection runs over the 10/100BaseT Ethernet interfaces of the controller.

CUL_SocketOpenUdpBind

The `CUL_SocketOpenUdpBind()`function creates a socket of UDP type and binds the socket to the selected port.

The binding address is always INADDR_ANY, i.e., all messages for UDP/port addressed to the COM module are received. Sockets are always run in non-blocking mode, i.e., this function does not block.

Function Prototype:

dword CUL_SocketOpenUdpBind(uword port, uword *assigned_port_ptr)

Parameter:

The function has the following parameters:

Parameter	Description
port	An available port number, not occupied by the COM >= 0. If the port parameter = 0, the socket is bound to the first available port.
assigned_port_ptr	Address to which the bounded port number should be copied, if the port parameter = 0 or NULL if not.

Table 287: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
Socket number	Socket number assigned to UDP if > 0 Error codes are < 0
CUL_ALREADY_BOUND	Impossible binding to a port for UDP
CUL_NO_MORE_SOCKETS	No more resources are available for socket
CUL_SOCK_ERROR	Other socket errors

Table 288: Return Value

Restrictions:

If the CUT does not possess assigned_port_ptr then CUT/CUIT are terminated.

CUL_SocketOpenUdp

The `CUL_SocketOpenUdp()`function creates a socket of UDP type without binding to a port. Afterwards, messages can only be sent, but not received via the socket.

Function Prototype:

dword CUL_SocketOpenUdp (void)

Parameter:

None

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
Socket number	Socket number assigned to UDP if > 0 Error codes are < 0
CUL_NO_MORE_SOCKETS	No more resources are available for socket
CUL_SOCK_ERROR	Other socket errors

Table 289: Return Value

CUL_NetMessageAlloc

The `CULMessageAlloc()`function allocates message memory for using

- `CUL_SocketSendTo()` with UDP and
- `CUL_SocketSend()` with TCP

A maximum of 10 messages can be simultaneously used in CUT.

Function Prototype:

void *CUL_NetMessageAlloc(udword size, ubyte proto)

Parameter:

The function has the following parameters:

Parameter	Description
size	Required memory size in bytes; it must be ≥ 1 byte and ≤ 1400 bytes
proto	0 = TCP 1 = UDP

Table 290: Parameter

Return Value:

Buffer address to which the data to be sent must be copied. Memory ranges must never be written outside the allocated area. There are no ranges for the used transmission protocols (EtherNet/IP/UDP or TCP).

Restrictions:

If no more memory resources are available or if the parameter size is too big or proto > 1, CUT and CUIT are terminated.

CUL_SocketSendTo

The `CUL_SocketSendTo()`function sends the message previously allocated and filled with the `CUL_NetMessageAlloc()`function as UDP package to the destIp/destPort target address.

After the message has been sent, the pMsg message memory is released automatically.

Whenever messages are sent, firstly the message memory must be allocated with the `CULMessageAlloc()`function.

Function Prototype:

```
dword CUL_SocketSendTo( dword socket,
                        void *pMsg,
                        udword size,
                        udword destIp,
                        uword destPort)
```

Parameter:

The function has the following parameters:

Parameter	Description
Socket	Socket created with CUL_SocketOpenUdp()
pMsg	UDP user data memory previously reserved with CUL_NetMessageAlloc()
Size	Memory size in bytes, it must be ≤ than the allocated memory
destIp	Target address != 0, also 0xffffffff is allowed as broadcast
destPort	Target port != 0

Table 291: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
CUL_OKAY	Message was sent successfully
CUL_NO_ROUTE	No routing available to obtain destIp
CUL_WRONG_SOCK	Wrong socket type or socket not available
CUL_SOCK_ERROR	Other socket errors

Table 292: Return Value

Restrictions:

If the CUT does not possess the pMsg message or if the size of pMsg is too high, CUT and CUIT are terminated.

CUCB_SocketUdpRcv

The COM invokes the CUCB_SocketUdpRcv() function if data from the socket is available. In callback, data must be copied from *pMsg to CUT data, if required. After the function return, no access to *pMsg is allowed.

Function Prototype:

```
void CUCB_SocketUdpRcv( dword socket,
                        void *pMsg,
                        udword packetLength,
                        udword dataLength)
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Socket created with CUL_SocketOpenUdp()
pMsg	pMsg points to the UDP package begin including the Ethernet header. The transmitter of the message can be identified via the Ethernet header.
packetLength	The length of the package is stored in packetLength, included the length of the header.
dataLength	The length of the UDP user data part is stored in dataLength.

Table 293: Parameter

CUL_NetMessageFree

The `CUL_NetMessageFree()`function releases the message previously allocated with `CUL_NetMessageAlloc()`.

This function is usually not required since invoking the `CUL_SocketSendTo()`function results in an automatic release.

Function Prototype:

void CUL_NetMessageFree(void *pMsg)

Parameter:

The function has the following parameter:

Parameter	Description
`pMsg`	Memory reserved by `CUL_NetMessageAlloc()`

Table 294: Parameter

Restrictions:

If the CUT does not possess the pMsg message, CUT and CUIT are terminated.

CUL_SocketOpenTcpServer

The `CUL_SocketOpenServer()`function creates a socket of type TCP and binds the socket to the selected port.

The address for binding is always INADDR_ANY. Additionally, the COM is requested to perform a *listen* on the stream socket. Sockets are always running in non-blocking mode, i.e., this function does not block.

For further information on how to use the socket, refer to `CUCB_SocketTryAccept()` and `CUL_SocketAccept()`.

Function Prototype:

dword CUL_SocketOpenTcpServer(uword port, udword backlog)

Parameter:

The function has the following parameters:

Parameter	Description
port	Port number not occupied by the COM > 0
backlog	Max. number of waiting connections for socket

Table 295: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
Socket number	Socket number already assigned to UDP if > 0 Error codes are < 0
CUL_ALREADY_BOUND	Binding to a port/proto not possible
CUL_NO_MORE_SOCKETS	No more resources are available for socket
CUL_SOCK_ERROR	Other socket errors

Table 296: Return Value

Restrictions:

If successfully one socket is used.

CUCB_SocketTryAccept

The COM invokes the `CUL_SocketTryAccept()`function if a TCP connection request is present.

This request can be used to create a socket with the `CUL_SocketAccept()`function.

Function Prototype:

void CUCB_SocketTryAccept(dword serverSocket)

Parameter:

The function has the following parameter:

Parameter	Description
serverSocket	Socket previously created by *CUL_SocketOpenTcpServer()*.

Table 297: Parameter

CUL_SocketAccept

The `CUL_SocketAccept()`function creates a new socket for the connection request previously signalized with *CUCB_SocketTryAccept()*.

Function Prototype:

```
dword CUL_SocketAccept( dword serverSocket,
                        udword *pIpAddr,
                        uword *pTcpPort)
```

Parameter:

The function has the following parameters:

Parameter	Description
serverSocket	serverSocket signalized with CUCB_SocketTryAccept()
pIpAddr	Address to which the IP address of the peer should be copied or 0 if not
pTcpPort	Address to which the TCP port number of the peer should be copied or 0 if not

Table 298: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
Socket	if > 0, new created socket
CUL_WRONG_SOCK	Wrong socket type or socket not available
CUL_NO_MORE_SOCKETS	No more socket resources are available
CUL_SOCK_ERROR	Other socket error

Table 299: Return Value

Restrictions:

If the CUT does not possess the messages pIpAddr and pTcpPort, CUT and CUIT are terminated.

CUL_SocketOpenTcpClient

The `CUL_SocketOpenTcpClient()`function creates a socket of type TCP with free local port and orders a connection to `destIp` and `destPort`. Sockets are always run in non-blocking mode, i.e., this function does not block. The `CUCB_SocketConnected()`function is invoked as soon as the connection has been established.

Function Prototype:

dword CUL_SocketOpenTcpClient(udword destIp, uword destPort)

Parameter:

The function has the following parameters:

Parameter	Description
destIp	IP address of the communication partners
destPort	Port number of the communication partners

Table 300: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
Socket number	if > 0; error codes are < 0
CUL_NO_MORE_SOCKETS	No more resources are available for socket
CUL_NO_ROUTE	No routing available to reach destIp
CUL_SOCK_ERROR	Other socket error

Table 301: Return Value

CUCB_SocketConnected

The `CUCB_SocketConnected()`function is invoked by the COM module if a TCP connection was established with the `CUL_SocketOpenTcpClient()`function.

Function Prototype:

void CUCB_SocketConnected(dword socket, bool successfully)

Parameter:

The function has the following parameters:

Parameter	Description
socket	Socket previously created and instructed by `CUL_SocketOpenTcpClient()`
successfully	TRUE if the connection attempt was successfully, otherwise FALSE

Table 302: Parameter

CUL_SocketSend

The `CUL_SocketSend()`function sends the message allocated and filled with the `CUL_NetMessageAlloc()`function as TCP package.

After the message has been sent, the pMsg message memory is released automatically.

Whenever messages are sent, firstly the message memory must be allocated with the `CULMessageAlloc()`function.

Function Prototype:

```
dword CUL_SocketSend(     dword socket,
                          void *pMsg,
                          udword size)
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Socket created by `CUL_SocketAccept()/CUL_SocketOpenTcpClient()`
pMsg	TCP user data memory previously reserved with CUL_NetMessageAlloc()
size	Memory size in bytes, it must be ≤ than the allocated memory

Table 303: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
CUL_OKAY	Message was sent successfully
CUL_WRONG_SOCK	Wrong socket type or socket not available
CUL_WOULD_BLOCK	Message cannot be sent, otherwise the socket would be blocked
CUL_SOCK_ERROR	Other socket error

Table 304: Return Value

Restrictions:

If the CUT does not possess the message pMsg or if the size of pMsg is too high, CUT and CUIT are terminated.

CUCB_SocketTcpRcv

The CUCB_SocketTcpRcv()function is invoked by the COM module if socket user data are placed.

After quitting the function CUCB_SocketTcpRcv(), *pMsg must no longer be accessed.

If user data are also required outside the CUCB_SocketTcpRcv()function, it must be copied from *pMsg in an area created ad hoc.

i If the TCP connection is closed asynchronously (after an error or due to a request from the other side), the CUCB_SocketTcpRcv()function with dataLength = 0 is selected.

The call signalized to CUT that the socket must be closed to re-synchronized communication.

Function Prototype:

```
void CUCB_SocketTcpRcv( dword socket,
                        void *pMsg,
                        udword dataLength)
```

Parameter:

The function has the following parameters:

Parameter	Description
socket	Socket used to receive the user data.
pMsg	The pMsg parameter points to the user data begin **without** Ethernet /IP /TCP header.
dataLength	Length of the user data in bytes

Table 305: Parameter

CUL_SocketClose

The `CUL_SocketClose()`function closes a socket previously created.

Function Prototype:

dword CUL_SocketClose(dword socket)

Parameter:

The function has the following parameter:

Parameter	Description
socket	Socket previously created

Table 306: Parameter

Return Value:

An error code (udword) is returned.

The error codes are defined in the cut.h header file.

Error Code	Description
CUL_OKAY	Socket closed and one socket resource free again.
CUL_WRONG_SOCK	Socket not available

Table 307: Return Value

11.5.8 Timer-IF

CUL_GetTimeStampMS

The `CUL_GetTimeStampMS()`function provides a millisecond tick. This tick is suitable for implementing an own timer in CUT/CUIT. The counter is derived from the quartz of the COM processor and has therefore the same precision.

Function Prototype:

udword CUL_GetTimeStampMS(void)

CUL_GetDateAndTime

The `CUL_GetDateAndTime()`function provides the seconds since the 1st January 1970, 00:00 to the *pSec memory and the milliseconds to the *pMsec memory. The values are compared with the safe CPU and, depending on the configuration, they can be externally synchronized via SNTP (see Chapter 9).

The values for `CUL_GetDateAndTime()`should **not** be used for time measurement, timer or the like since they can be provided by the synchronization or by the user during operation.

Function Prototype:

void CUL_GetDateAndTime(udword *pSec, udword *pMsec)

Restrictions:

If the memory of pSec or pMsec are not allocated in the CUT data segment, CUT and CUIT are terminated.

11.5.9 Diagnosis

The `CUL_DiagEntry()`function records an event in the COM short time diagnosis that can be read out using the PADT.

Function Prototype:

```
void CUL_DiagEntry(    udword severity,
                       udword code,
                       udword param1,
                       udword param2)
```

Parameter:

The function has the following parameters:

Parameter	Description
severity	It is sued to classify events 0x45 ('E') == error, 0x57 ('W') == warning, 0x49 ('I') == information
code	The user defines the parameter code with an arbitrary number for the corresponding events. If the event occurs, the number is displayed in the diagnosis.
param1, param2	Additional information on the event

Table 308: Parameter

11.6 Installing the Development Environment

This chapter describes how to install the development environment and create a ComUserTask.

The development environment is contained in the installation CD (see also chapter 11.2).

11.6.1 Installing the Cygwin Environment

The Cygwin environment is required since the GNU C compiler tools only runs under the Cygwin environment.

Cygwin environment must be installed under Windows 2000/XP/Vista.

i Observe the installation requirements described in Chapter 11.2. **Deactivate the virus scanner** on the PC on which Cygwin should be installed to avoid problems when installing Cygwin.

Perform the following steps to install the Cygwin environment:

Start the setup program for installing Cygwin:

1. Copy the Cygwin installation archive cygwin_2005-03-11 from the installation CD to the local hard disk (e.g., `drive C:\`).
2. Open the Cygwin index in Windows Explorer `C:\ cygwin_2005-03-11`.
3. Double-click the **setup.exe** file to start the Cygwin installation.
4. Click the **Next** button to start the setup.

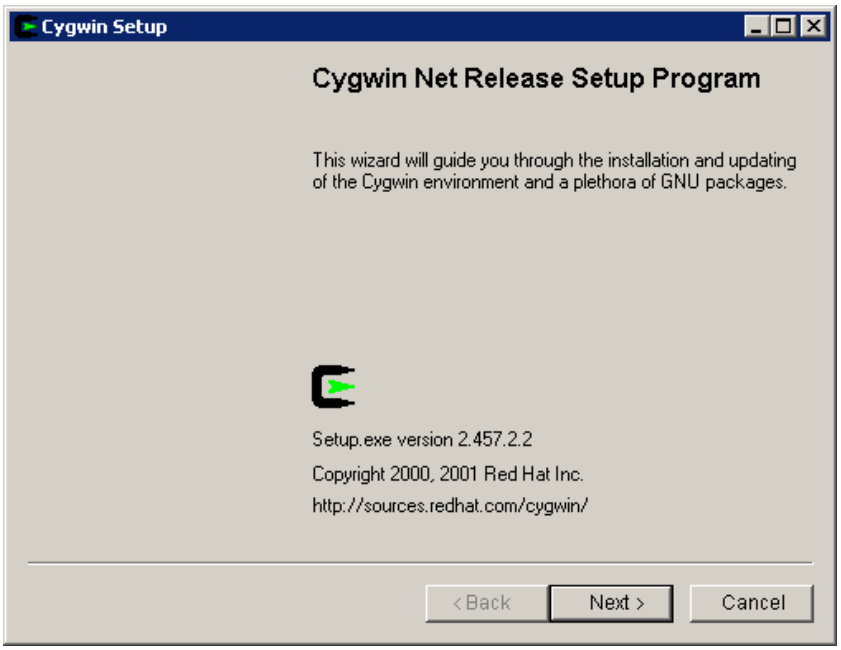


Figure 83: *Cygwin Setup* Dialog Box

The Disable Virus Scanner dialog box appears if the virus scanner was not deactivated.

Follow these steps to deactivate the virus scanner during the installation of Cygwin.

i Deactivate the virus scanner before installing Cygwin since, depending on the virus scanner in use, the warning dialog box could not appear although the virus scanner is running.

1. Select **Disable Virus Scanner** to prevent potential problems during the installation due to the virus scanner.
2. Click the **Next** button to confirm.

Select the installation source in the *Choose Installation Type* dialog box:

1. Select **Install from Local Directory** as installation source.
2. Click the **Next** button to confirm.

In the Choose Installation Directory dialog box, select the installation target directory for Cygwin:

1. Enter the directory in which Cygwin should be installed.
2. Accept all the defaults of the dialog box.
3. Click the **Next** button to confirm.

In the Select Local Package Directory dialog box, select the Cygwin installation archive.

1. In the *Local Package Directory* field, enter the name of the Cygwin installation archive containing the installation files
2. Click the **Next** button to confirm.

In the Select Packages dialog box, select all installation packages:

1. Select the **Curr** radio button.
2. In the view box, slowly click the installation option next to **All** until **Install** is displayed for a complete installation of all packages (approx. 1.86 GB memory requirements).

i Make sure that Install is placed behind each package.
If the packages are not completely installed, important functions might be missing for compiling the CUT C code!

3. Click the **Next** button to confirm.

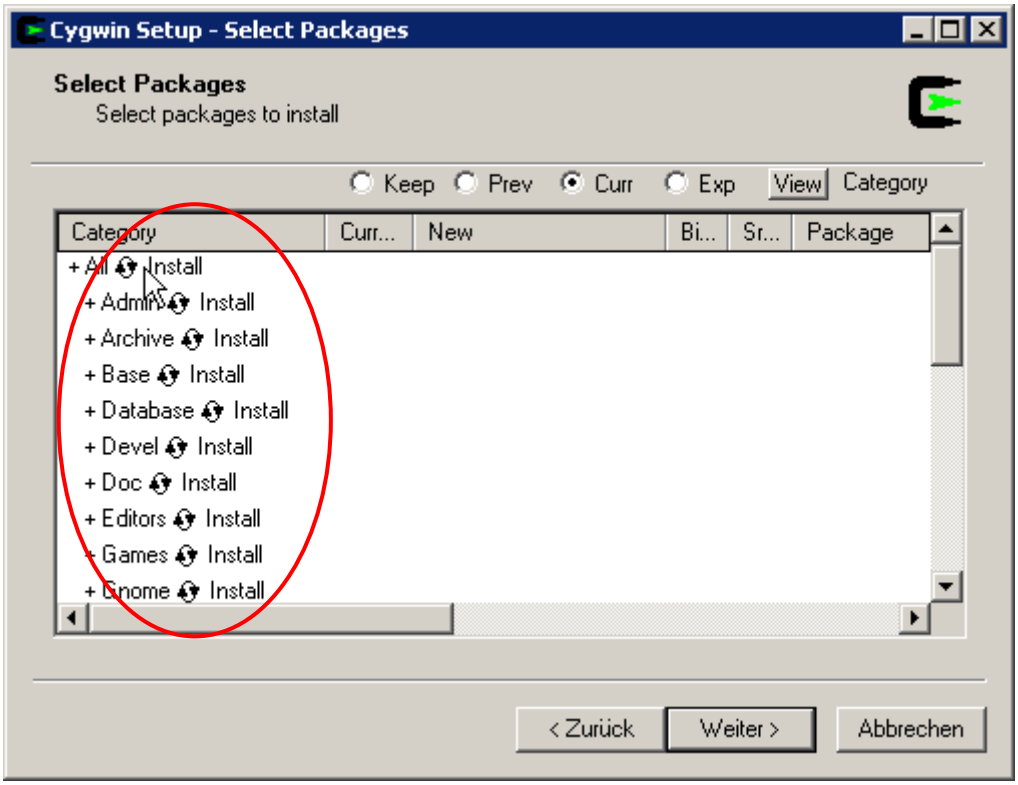


Figure 84: *Select Packages* Cygwin Setup Dialog Box

Perform the following steps to complete the Cygwin installation:

1. Select Entry in the **Start Menu**.
2. Select **Desktop Icon**.
3. Click the **Finish** button to complete the Cygwin installation.

Cygwin Commands	Description
cd (directory name)	Change directory
cd ..	Move to parent directory
ls -l	Display all files of a directory
help	Overview of bash shell commands

Table 309: Commands in Cygwin (Bash Shell)

11.6.2 Installing the GNU Compiler

Perform the following steps to install the GNU Compiler:

1. In Windows Explorer, open the directory of the installation CD.
2. Double-click `gcc-ppc-v3.3.2_binutils-v2.15.zip`.
3. Extract all files in the Cygwin directory (e.g. C:\cygwin\…).
   The GNU compiler is unpacked in the **gcc-ppc** subdirectory.
4. Set the environment variables in the system control:

- Use the Windows start menu **Settings->System Control->System** to open the system properties.
- Select the **Advanced** tab.
- Click the **Environment Variables** button.
- Select the **Path** system variable in the *System Variables* box and extend the system variable with `C:\cygwin\gcc-ppc\bin`.

Copy the cut_src folder from the installation CD to the home directory.
The cut_src folder contains all the "include" and "lib" directories required for creating a ComUserTask. The cut_cbf.c source file is located by default in ...\cut_src\cutapp\ directory.

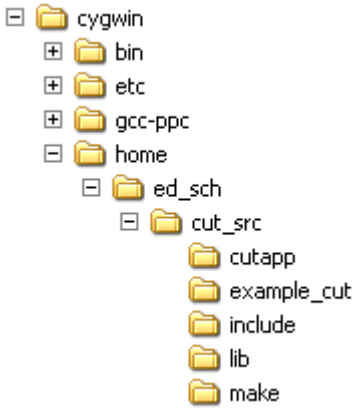


Figure 85: Cygwin Structure Tree

i If the home directory was not created automatically, create it with Windows Explorer (e.g., `C:\cygwin\home\User1`).

To create another home directory for cygwin bash shell, add the `set Home` command to the cygwin.bat batch file.

@echo off

C:

chdir C:\cygwin\bin

set Home=C:\User1

bash --login -i

Figure 86: *Cygwin.bat* Batch File

11.7 Creating New CUT Projects

This chapter shows how to create a new CUT project and specifies which files must be adapted.

i The **example cut** CUT project located on the installation CD is fully adapted.

For creating new CUT projects, HIMA recommends to create a new subdirectory of *...\cut_src* for each CUT project.

Example:

As a test, create the *example cut* directory, name the C source **example_cut.c**, the ldb file created in the *make* directory is named **example_cut.ldb**.

Create the folder example_cut for the new ComUserTask.

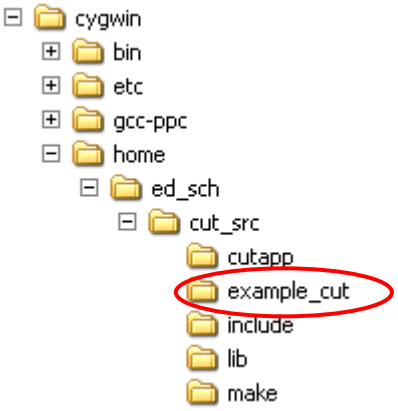


Figure 87: Cygwin Structure Tree

1. Copy the files
   - cutapp.c
   - cutapp.mke
   - makefile

   from the directory *cutapp* in the directory *example_cut*.
2. The files must be renamed using loadable names (e.g., **cutapp** in example_cut).

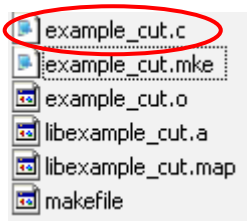


Figure 88: C Code File in the example_cut Folder

As described in the following chapters, perform all changes to .mke file and makefile for every new project.

11.7.1 CUT Makefiles

Configuration of CUT makefiles for different source files and ldb files.

As described in the following chapters, a total of three makefiles must be adapted.

Makefile with ".mke" Extension

The .mke file is located in the corresponding source code directory, e.g., *cut_src\example_cut\example_cut.mke.*

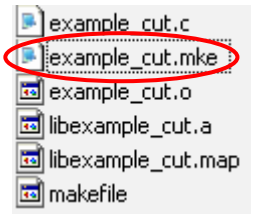


Figure 89: .mke File in the example_cut Folder

Change the mke file as described below:

1. The **module** variable must have the same loadable name as the corresponding .mke file (e.g., example_cut).
2. One or several C files required for creating the target code (loadable file) are assigned to the **c_sources** variable .

```
##############################################################################
#
# make file (DOS/NT)
# $Id: cutapp.mke 58869 2005-10-11 12:35:46Z es_fp $
##############################################################################
#
# assign name of module here (e.g. nl for NetworkLayer)
module= example_cut
#
# assign module sources here
sources=

c_sources= $(module).c
asm_sources=
```

Figure 90: .mke File Starting with Line 1

Makefile

The makefile file is located in the corresponding source code directory, e.g., `cut_src\example_cut\makefile.`

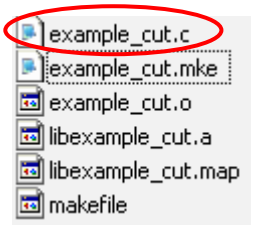


Figure 91: makefile in the *example_cut* Folder

Change the makefile file as described below:

1. Drag the include line for the .mke file upwards and enter the current name for the the .mke file.
2. Expand the make call with the two variables **SUBMOD_DIRS** and **CUT_NAME.**

```
all_objects:

include example_cut.mke

cut:
$(MAKE) -C ../make elf SUBMOD_DIRS=cut_src/$(module) CUT_NAME=$(module)


all_objects: $(c_objects) $(asm_objects) $(objects)
```

Figure 92: makefile Starting with Line 34

Makefile with the "makeinc.inc.app" Extension

The only change made to this and all the following CUT projects is that the name of the CUT loadable is made changeable via a make variable.

The makeinc.inc.app file is located in the cut_src directory e.g., cut_src\makeinc.inc.app.

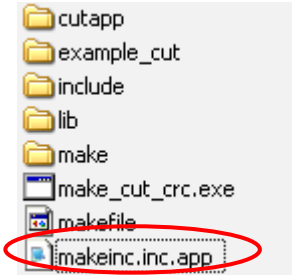


Figure 93: makeinc.inc.app file in the *example_cut* Folder

Change the makeinc.inc.app file as described below:

1. Expand the file with the **CUT_NAME** variable.

```
all: lib$(module).$(LIBEXT)
@echo 'did make for module ['lib$(module).$(LIBEXT)']'

lib$(module).$(LIBEXT): $(objects) $(c_objects) $(asm_objects) $(libraries)

SUBMOD2_LIBS=$(foreach lib,$(SUBMOD_LIBS),../../$(lib))

CUT_NAME=cut

makeAllLibs:
$(MAKE) -C ../../cut_src cut_src


makeLoadable:
@echo; \
BGTYPE=" $(CUT_NAME)"; \
if [ ! -f $$BGTYPE.map ] ; then \
echo "Error: MAP-Datei $$BGTYPE.map existiert nicht"; \
exit 1; \
fi; \
OS_LENGTH=$$(gawk '/___OS_LENGTH/ {print substr($$1,3,8)}' $$BGTYPE.map); \
echo; \
$(OBJCOPY) --strip-all --strip-debug -O binary $$BGTYPE.elf $$BGTYPE.bin;\
echo; \
echo "Building C3-Loadable-Binary ..."; \
$(MCRC) $$BGTYPE.bin 0 $$OS_LENGTH $$OS_LENGTH $$BGTYPE.ldb; \
echo; \

$(CUT_NAME).elf:  makeAllLibs $(SUBMOD2_LIBS)

elf:
@echo; test -f section.dld && $(MAKE) $(CUT_NAME).elf && $(MAKE) makeLoadable \
|| { echo "ERROR: Wrong subdir. Please invoke elf target only from make/ subdirectory."
&& echo && false ; } ;

# end of file: makeinc.inc
```

Figure 94: makeinc.inc.app Starting with Line 247

11.7.2 Adapting C Source Codes

Perform the following steps to open the source code file:

1. Open the project directory *cut_src**example_cut* that has been created and configured in the previous steps.
2. Use an editor (e.g., Notepad) to open the C source code file with the extension **.c**.

Configure Input and Output Variables

Perform the following steps to configure the input and output variables in the source code file:

1. The data size of the variables that should be created in the SILworX **Output Variables** tab must be set in the **CUT_PDI[X]** array of the source code file.
2. The data size of the variables that should be created in the SILworX **Data Inputs** tab must be set in the **CUT_PDO[X]** array of the source code file.

Start Function in CUT

The `void CUCB_TaskLoop (udword mode)C` function is the start function and is cyclically invoked by the user program.

Example Code "example_cut.c"

The following C code copies the value from the **CUT_PDI[0]** input to the **CUT_PDO[0]** output and returns the value unchanged to the SILworX user program.

Note	The C code **example_cut.c** is located on the installation CD.

```
/* Example for the CUT implemetation */
#include "include/cut_types.h"
#include "include/cut.h"
#ifdef __cplusplus
extern "C" {
#endif
/*****************************************************************************/
/* SILworX Output Records (CPU->COM) */
uword CUT_PDI[1] __attribute__ ((section("CUT_PD_IN_SECT"), aligned(1)));
/* SILworX Input Records (COM->CPU) */
uword CUT_PDO[1] __attribute__ ((section("CUT_PD_OUT_SECT"), aligned(1)));
/*****************************************************************************/

/* Callback function for starting the CUT */
  void CUCB_TaskLoop(udword mode)
  {
    if (CUT_PDI[0] > CUT_PDO[0])/*This is executed only, if the          */
      {                         /*SILworX application program    */
                                /*was processed.                       */
                                /*The SILworX application program*/
                                /*adds the value 1 to CUT_PDO[0] and   */
                                /*writes the result into CUT_PDI[0]    */

        CUT_PDO[0] = CUT_PDI[0];/*Copies the value from input CUT_PDI[0] */
                                /*into output CUT_PDO[0] of the SPS    */
        if (CUT_PDO[0] == 65535)
           {CUT_PDO[0] = 0;}
      }
  }
/*****************************************************************************/
```

Figure 95: Resource Structure Tree

```
/*****************************************************************************/
  void CUCB_AscRcvReady(udword comId)
  {
    CUL_DiagEntry(0x49, 1, comId, 0);
  }
/*****************************************************************************/
  void CUCB_AscSendReady(udword comId)
  {
    CUL_DiagEntry(0x49, 2, comId, 0);
  }
/*****************************************************************************/
  void CUCB_SocketTryAccept(dword serverSocket)
  {
    CUL_DiagEntry(0x49, 3, serverSocket, 0);
  }
/*****************************************************************************/
  void CUCB_SocketConnected(dword socket, bool Okay)
  {
    CUL_DiagEntry(0x49, 4, socket, Okay);
  }
/*****************************************************************************/
  void CUCB_SocketTcpRcv(dword socket, void *pMsg, udword dataLength)
  {
    CUL_DiagEntry(0x49, 5, socket, dataLength);
  }
/*****************************************************************************/
  void CUCB_SocketUdpRcv(dword socket, void *pMsg, udword packetLength,
                         udword dataLength)
  {
    CUL_DiagEntry(0x49, 6, socket, dataLength);
  }
/*****************************************************************************/
  void CUCB_IrqService(udword devNo)
  {
    CUL_DiagEntry(0x49, 7, devNo, 0);
  }
/*****************************************************************************/

#ifdef __cplusplus
} /* end extern "C" */
#endif

/* end of file */
```

Figure 96: C Code example_cut.c

Creating Executable Codes (ldb file)

Perform the following steps to create the executable code (ldb file):

1. Start **Cygwin Bash Shell.**
2. Move to the directory */cut_src/example_cut/.*
3. Use the command `make cut` to start the code generation.
   The **cut.ldb** binary file located in the */cut_src/make/* directory is generated automatically.
4. If CRC32 was generated, also the executable code was generated (see red marking in Figure 97).

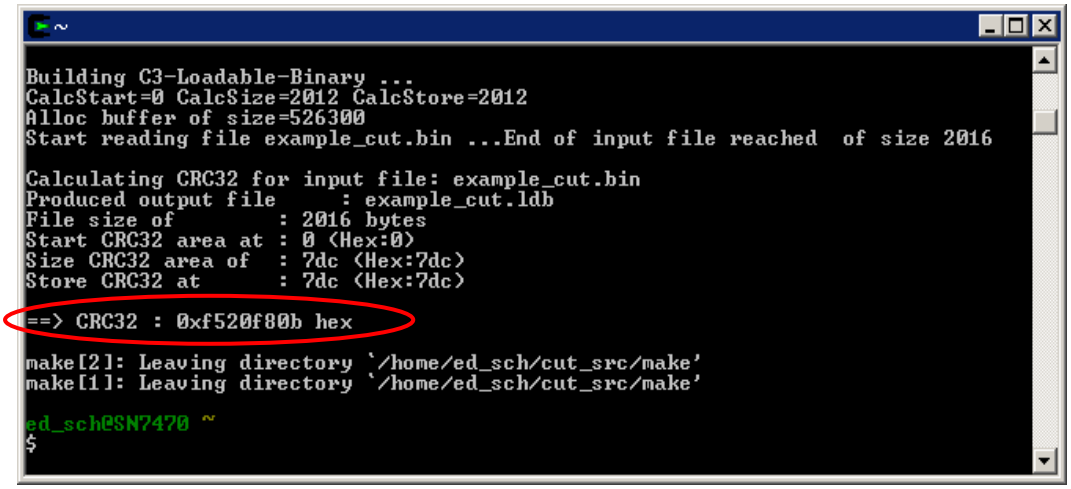


Figure 97: Cygwin Bash Shell

This executable code (ldb file) must eb loaded in the ComUserTask, see Chapter 11.7.3.

11.7.3 Implementing the ComUserTask in the Project

Perform the following steps in SILworX to integrate the ComUserTask into the project:

Creating the ComUserTask

To create a new ComUserTask

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. On the context menu for protocols, click **New, ComUserTask** to add a new ComUserTask.
3. Right-click the ComUserTask, click **Properties** and select the **COM Module**. Accept the default settings for the first configuration.

---

i Only one ComUserTask per resource may be created.

Loading Program Code into the Project

To load a ComUserTask into the project

1. In the structure tree, open **Configuration, Resource, Protocols**.
2. Right-click **ComUserTask**, then click **Load User Task**. Open the directory *.../cut_src/make/*
3. Select the **ldb file** that should be processed in the ComUserTask.

---

i Different versions of the ldb file can be integrated by reloading the executable code (ldb file). The correctness of the ldb file's content is not checked when loading. The ldb file is then compiled in the project together with the resource configuration and can be loaded into the controller. If the ldb file is changed, the project must be recompiled and reloaded.

Connecting Variables to CUT

The user can define a not safety-related process communication between the safe CPU and the not safe COM (CUT). A maximum of 16-kByte data can be exchanged per direction.

Create the following global variables:

Variable	Type
COM_CPU	UINT
CPU_COM	UINT

Connecting Process Variables

Process Variables in the ComUserTask

1. Right-click **ComUserTask**, then click **Edit**.
2. In the **Edit** dialog box, select the **Process Variables** tab.

Output Variables (CPU->COM)

The **Output Variables** tab contains the variables that should be transferred from the CPU to the COM module.

Name	Type	Offset	Global Variable
CPU_COM	UINT	0	CPU_COM

Table 310: Output Variables (CPU->COM)

1. Drag the global variables to be sent from the Object Panel onto the **Output Variables** tab.
2. Right-click anywhere in the **Output Variables** area to open the context menu.
3. Click **New Offsets** to re-generate the variable offsets.

Input Variables (COM->CPU)

The **Input Variables** tab contains the variables that should be transferred from the COM to the CPU module.

Name	Type	Offset	Global Variable
COM_CPU	UINT	0	COM_CPU

Table 311: Input Variables(COM->CPU)

1. Drag the global variables to be received from the Object Panel onto the **Input Variables** area.
2. Right-click anywhere in the the **Input Variables** area to open the context menu.
3. Click **New Offsets** to re-generate the variable offsets.

To verify the ComUserTask configuration

1. In the structure tree, select **Configuration, Resource, Protocols, ComUserTask**.
2. Right-click **Verification** to verify the CUT configuration.
3. Thoroughly verify the messages contained in the logbook and correct potential errors.

Creating the SILworX User Program

To create the SILworX user program

1. In the structure tree, open **Configuration,** right-click **Resource**, then select **Edit**.
2. Drag the global variables **COM_CPU** and **CPU_COM** from the Object Panel to the drawing area.
3. Create the user program as specified in the following figure.

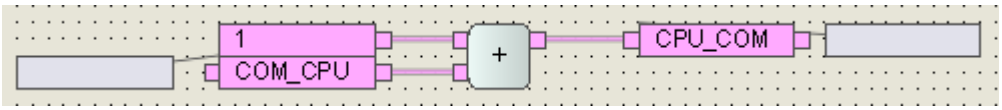


Figure98: SILworX Program Editor

To configure the schedule interval [ms]

1. Right-click **ComUserTask,** then click **Properties**.
2. In the *Schedule Interval [ms]* input box, specify in which intervals the ComUserTask should be invoked.

---

i Use the resource's user program to recompile the ComUserTask configuration and load it into the controller. The new configuration can only be used for communicating with the HIMax system after this step is completed

---

To check the ComUserTask with the online test

1. In the structure tree, select C**onfiguration, Resource, Program.**
2. Right-click **Program** and select **Online**. Log in to the system.

I

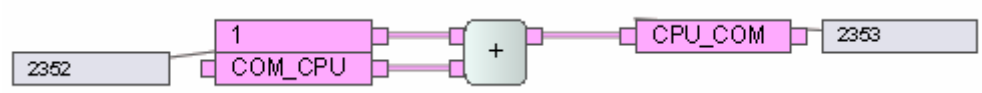


Figure 99: SILworX Online Test

Function of the SILworX user program:

The SILworX user program adds the value **1** to the signal **COM_CPU** (data inputs) and transmits the result to the signal **CPU_COM** (data outputs).

With the next CUT call (schedule interval [ms]), the signal **CPU_COM** is transmitted to the CUT function (see example code in Chapter 11.7.2).

The ComUserTask receives the signal **CPU_COM** and returns the value unchanged with the signal **COM_CPU**.

11.7.4 Faults while Loading a Configuration with CUT

Run Time Problems (e.g., ComUserTask in infinite loop):

Reason for run time problems:

Programming a loop, which runs for a long time, in the corresponding CUT source code results in a “deadlock“ of the COM processor.

As a consequence, no connection can be established to the controller and the resource configuration can no longer be deleted.

Solution: Reset the HIMax communication module or the HIMatrix controller:

- In the Online View associated with the Hardware Editor, use the **Maintenance/Service, Module (Restart)** function to reset the communication module (or the reset push-button to reset the HIMatrix system, see the controller's data sheet).
- Create a new CUT (without run time errors, endless loops).
- Load the CUT (ldb file) into the project.
- Generate the code.
- Load the code into the controller.

12 General

12.1 Configuring the Function Blocks

The fieldbus protocols and the corresponding function blocks operate on the COM module of the HIMax controller. Therefore, the function blocks must be created in the SILworX. To do this, open **Configuration, Resource, Protocols...** in the structure tree.

To control the function blocks on the COM module, function blocks can be created in the SILworX user program (see Chapter 12.1.1). These can be used as standard function blocks.

Common variables are used to connect the function blocks in the SILworX user program to the corresponding function blocks in the SILworX structure tree. These must be created beforehand using the Variable Editor.

12.1.1 Purchasing Function Block Libraries

The function block libraries for PROFIBUS DP and TCP Send/Receive must be added to the project using the *Restore...* function (from the context menu for the project).

The function block libraries are available upon request from HIMA support.

Tel: +49 (0)6202-709 -185

-259

-261

E-mail: support@hima.com

12.1.2 Configuring the Function Blocks in the User Program

Drag the required function blocks onto the user program. Configure the inputs and outputs as described for the individual function block.

Upper Part of the Function Block

The upper part of the function block corresponds to the user interface that the user program uses for controlling it.

The variables used in the user program are connected here. The prefix "A" means Application.

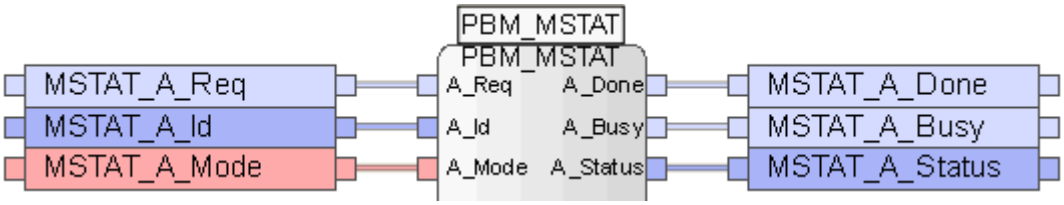


Figure 100: PNM_MSTST Function Block (Upper Part)

Lower Part of the Function Block

The function block's lower part represents the connection to the function block (in the SILworX structure tree).

The variables that must be connected to the function block in the SILworX structure tree are connected here. The prefix "F" means "Field".

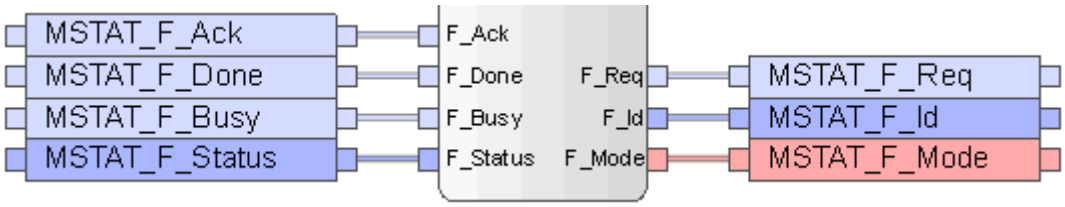


Figure 101: PNM_MSTST Function Block (Lower Part)

12.1.3 Configuring the Function Blocks in the SILworX Structure Tree

To configure the function block in the SILworX structure tree

1. In the structure tree, open **Configuration, Resource, Protocols**, e.g., **PROFIBUS Master**.
2. Right-click **Function Blocks** , and then click **New**.
3. In the SILworX structure tree, click the suitable function block.

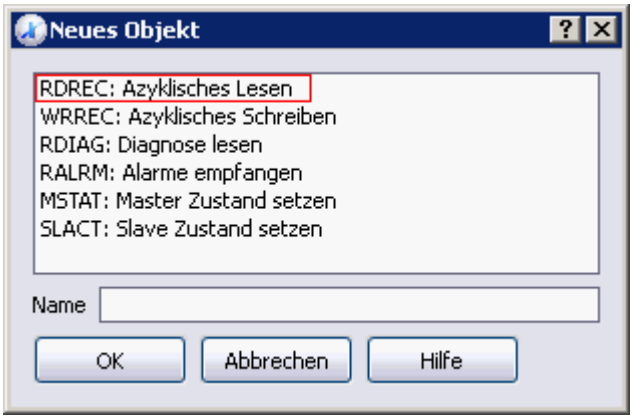


Figure 102: Choosing Function Blocks

The inputs of the function block (checkmark in the Input Variables column) must be connected to the same variables that are connected in the user program to the *F_Outputs* of the function block.

The outputs of the function block (no checkmark in the Input Variables column) must be connected to the same variables that are connected in the user program to the *F_Inputs* of the function block.

Systemvariablen

F	Name	Datentyp	Transfer-Operation	Eingangsvariable	Globale Variable
1	ACK	BOOL	5	☑	MSTAT_F_Ack
2	BUSY	BOOL	5	☑	MSTAT_F_Busy
3	DONE	BOOL	5	☑	MSTAT_F_Done
4	REQ	BOOL	5	☐	MSTAT_F_Req
5	M_ID	DWORD	5	☐	MSTAT_F_Id
6	STATUS	DWORD	5	☑	MSTAT_F_Status
7	MODE	INT	5	☐	MSTAT_F_Mode

Figure 103: System Variables of the MSTAT Function Block

12.2 Maximum Communication Time Slice

The maximum communication time slice is the time period in milliseconds (ms) und per CPU cycle assigned to the processor module for processing the communication tasks.
If not all upcoming communication tasks can be processed within one CPU cycle, the whole communication data is transferred over multiple CPU cycles (number of communication time slices > 1).

To determine the maximum communication tiem slice

1. All communication protocols are operating (safe**ethernet** and standard protocols).
2. Open the **Control Panel** and select the **Statistics** directory in the structure tree.
3. Open **Cyc.Async** to read the number of communication time slices.
4. Open **Time.Async** to read the duration of one communication time slice.

i The number of communication time slices must be = 1 when the maximum reaction times allowed is calculated (see Chapter 4.7).

The duration of the communication time slice must be set such that the CPU cycle using the communication time slice cannot exceed the watchdog time specified by the process.

Appendix

Glossary

Term	Description
ARP	Address Resolution Protocol: Network protocol for assigning the network addresses to hardware addresses
AI	Analog Input
Connector board	Connector board for the HIMax module
COM	Communication module
CRC	Cyclic Redundancy Check
DI	Digital Input
DO	Digital Output
EMC	Electromagnetic Compatibility
EN	European Norm
ESD	ElectroStatic Discharge
FB	Fieldbus
FBD	Function Block Diagram
FTA	Field Termination Assembly
FTT	Fault Tolerance Time
ICMP	Internet Control Message Protocol: Network protocol for status or error messages
IEC	International Electrotechnical Commission
MAC Address	Hardware address of one network connection (Media Access Control)
PADT	Programming And Debugging Tool (in accordance with IEC 61131-3), PC with SILworX
PE	Protective Earth
PELV	Protective Extra Low Voltage
PES	Programmable Electronic System
PFD	Probability of Failure on Demand, probability of failure on demand of a safety function
PFH	Probability of Failure per Hour, probability of a dangerous failure per hour
R	Read
Rack ID	Base plate identification (number)
Non-reactive	Supposing that two input circuits are connected to the same source (e.g., a transmitter). An input circuit is termed "non-reactive" if it does not distort the signals of the other input circuit.
R/W	Read/Write
SB	System Bus (Module)
SELV	Safety Extra Low Voltage
SFF	Safe Failure Fraction, portion of safely manageable faults
SIL	Safety Integrity Level (in accordance with IEC 61508)
SILworX	Programming tool for HIMax
SNTP	Simple Network Time Protocol (RFC 1769)
SRS	System.Rack.Slot
SW	Software
TMO	Timeout
W	Write
WD	Watchdog
WDT	Watchdog Time

Index of Figures

Index of Tables

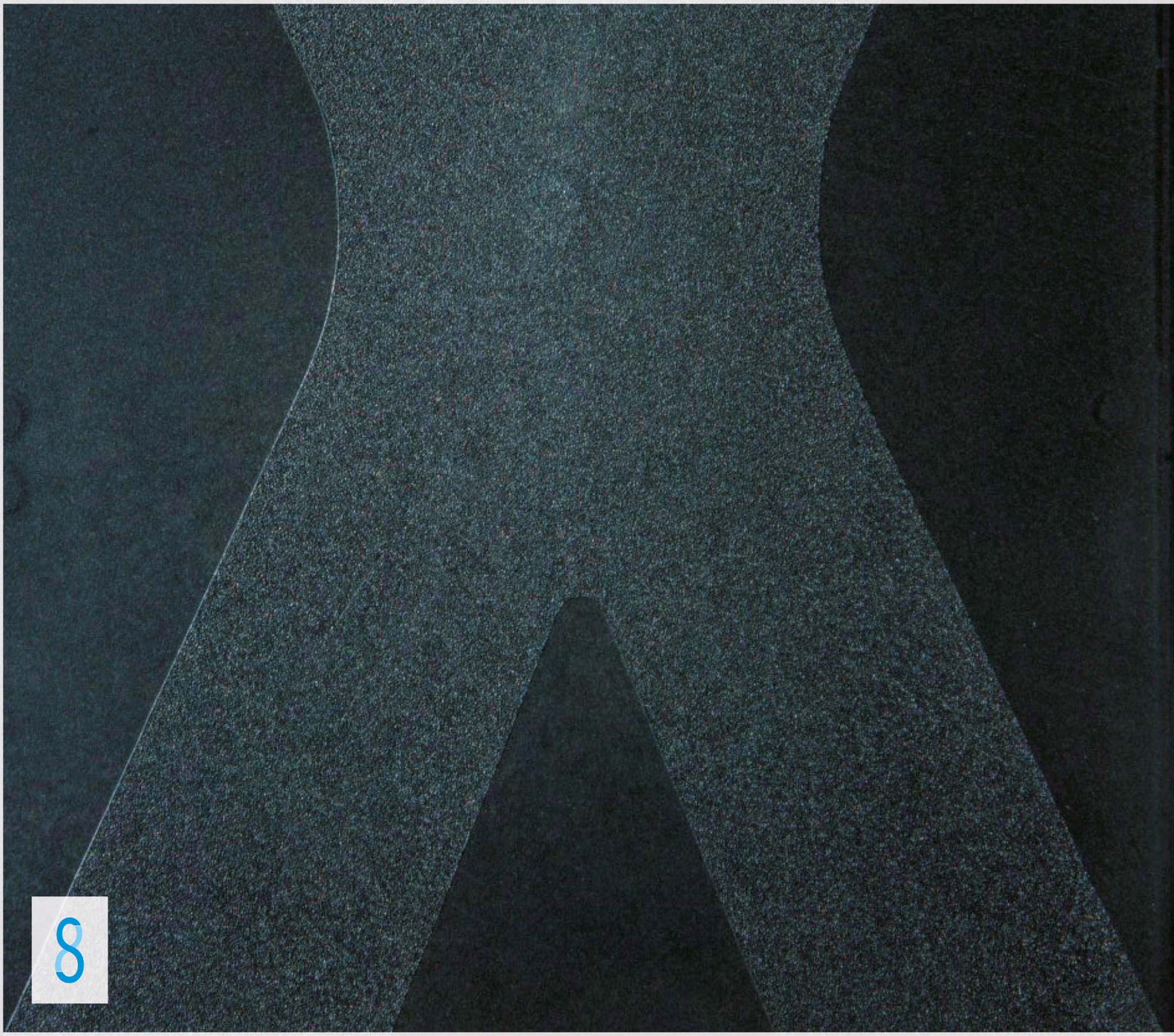
Index

HI 801 101 E

Albert-Bassermann-Str. 28
68782 Brühl, Deutschland
Tel. +49 6202 709-0
Fax +49 6202 709-107
HIMax-info@hima.com
www.hima.com

HIMA

SAFETY
NONSTOP



8